AccessWare
User Guide

Optegra® Release 6

DOC40184-006

Parametric Technology Corporation

Copyright © 2001 Parametric Technology Corporation. All Rights Reserved.
User documentation from Parametric Technology Corporation (PTC) is subject to copyright laws of the United States and other countries and is provided under a license agreement, which restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed user the right to make copies in printed form of PTC user documentation provided on software or documentation media, but only for internal, noncommercial use by the licensed user in accordance with the license agreement under which the applicable software and documentation are licensed. Any copy made hereunder shall include the Parametric Technology Corporation copyright notice and any other proprietary notice provided by PTC. User documentation may not be disclosed, transferred, or modified without the prior written consent of PTC and no authorization is granted to make copies for such purposes.

Information described in this document is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION.

Registered Trademarks of Parametric Technology Corporation or a Subsidiary
Advanced Surface Design, CADDS, CADDShade, Computervision, Computervision Services, Electronic Product Definition, EPD, HARNESSDESIGN, Info*Engine, InPart, MEDUSA, Optegra, Parametric Technology, Parametric Technology Corporation, Pro/ENGINEER, Pro/HELP, Pro/INTRALINK, Pro/MECHANICA, Pro/TOOLKIT, PTC, PT/Products, Windchill, InPart logo, and PTC logo.

Trademarks of Parametric Technology Corporation or a Subsidiary
3DPAINT, Associative Topology Bus, Behavioral Modeler, BOMBOT, CDRS, CounterPart, CV, CVact, CVaec, CVdesign, CV-DORS, CVMAC, CVNC, CVToolmaker, DesignSuite, DIMENSION III, DIVISION, DVSAFEWORK, DVS, e-Series, EDE, e/ENGINEER, Electrical Design Entry, Expert Machinist, Expert Toolmaker, Flexible Engineering, *i*-Series, ICEM, Import Data Doctor, Information for Innovation, ISSM, MEDEA, ModelCHECK, NC Builder, Nitidus, PARTBOT, PartSpeak, Pro/ANIMATE, Pro/ASSEMBLY, Pro/CABLING, Pro/CASTING, Pro/CDT, Pro/CMM, Pro/COMPOSITE, Pro/CONVERT, Pro/DATA for PDGS, Pro/DESIGNER, Pro/DESKTOP, Pro/DETAIL, Pro/DIAGRAM, Pro/DIEFACE, Pro/DRAW, Pro/ECAD, Pro/ENGINE, Pro/FEATURE, Pro/FEM-POST, Pro/FLY-THROUGH, Pro/HARNESS-MFG, Pro/INTERFACE for CADDS 5, Pro/INTERFACE for CATIA, Pro/LANGUAGE, Pro/LEGACY, Pro/LIBRARYACCESS, Pro/MESH, Pro/Model.View, Pro/MOLDESIGN, Pro/NC-ADVANCED, Pro/NC-CHECK, Pro/NC-MILL, Pro/NC-SHEETMETAL, Pro/NC-TURN, Pro/NC-WEDM, Pro/NC-Wire EDM, Pro/NCPOST, Pro/NETWORK ANIMATOR, Pro/NOTEBOOK, Pro/PDM, Pro/PHOTORENDER, Pro/PHOTORENDER TEXTURE LIBRARY, Pro/PIPING, Pro/PLASTIC ADVISOR, Pro/PLOT, Pro/POWER DESIGN, Pro/PROCESS, Pro/REPORT, Pro/REVIEW, Pro/SCAN-TOOLS, Pro/SHEETMETAL, Pro/SURFACE, Pro/VERIFY, Pro/Web.Link, Pro/Web.Publish, Pro/WELDING, Product Structure Navigator, PTC *i*-Series, Shaping Innovation, Shrinkwrap, The Product Development Company, Virtual Design Environment, Windchill e-Series, CV-Computervision logo, DIVISION logo, and ICEM logo.

Third-Party Trademarks
Oracle is a registered trademark of Oracle Corporation. Windows and Windows NT are registered trademarks of Microsoft Corporation. Java and all Java based marks are trademarks or registered trademarks of Sun Microsystems, Inc. CATIA is a registered trademark of Dassault Systems. PDGS is a registered trademark of Ford Motor Company. SAP and R/3 are registered trademarks of SAP AG Germany. FLEX*lm* is a registered trademark of GLOBEtrotter Software, Inc. VisTools library is copyrighted software of Visual Kinematics, Inc. (VKI) containing confidential trade secret information belonging to VKI. HOOPS graphics system is a proprietary software product of, and copyrighted by, Tech Soft America, Inc. All other brand or product names are trademarks or registered trademarks of their respective holders.

UNITED STATES GOVERNMENT RESTRICTED RIGHTS LEGEND
This document and the software described herein are Commercial Computer Documentation and Software, pursuant to FAR 12.212(a)-(b) or DFARS 227.7202-1(a) and 227.7202-3(a), and are provided to the Government under a limited commercial license only. For procurements predating the above clauses, use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or Commercial Computer Software-Restricted Rights at FAR 52.227-19, as applicable.

Parametric Technology Corporation, 140 Kendrick Street, Needham, MA 02494-2714
8 January 2001

Table of Contents

Preface

Related Documents ____ xv
Book Conventions ____ xvi
Online User Documentation ____ xvii
Printing Documentation ____ xvii
Resources and Services ____ xviii
Documentation Comments ____ xviii

Introduction to AccessWare

Introduction ____ 1-2
Using AccessWare to Create a GUI ____ 1-2
Basic Concepts of AccessWare ____ 1-4
Resource Files ____ 1-4
Application Actions ____ 1-5
Color, Icons and Help ____ 1-5
Sample Resource File ____ 1-6

AccessWare Setup Procedures

Understanding AccessWare Installation ____ 2-2
AccessWare System Contents ____ 2-2
Setting the Development Environment ____ 2-3
Setting the Runtime Environment ____ 2-3

Setting Environment Variables ____ 2-4
Summary of the Environment Variables ____ 2-4
AW_RESOURCE ____ 2-5
Colors as Environment Variables ____ 2-6
AW_PROMPT_COLOR ____ 2-6
AW_MESSAGE_COLOR ____ 2-6
AW_TEXT_COLOR ____ 2-7
AW_HILITE_COLOR and AW_LOLITE_COLOR ____ 2-7
AW_SCROLL_FACTOR ____ 2-7
AW_MESSAGE_FILE ____ 2-8
AW_XICON_TYPE ____ 2-8
AW_WINDOW_STATE ____ 2-8
ACCESSTRACE ____ 2-8
CMOM_DOMAIN ____ 2-9
CMOM_DISPLAY ____ 2-9
AW_QUERY_ON_LIST and AW_QUERY_ON_TEXT ____ 2-9
Executing an AccessWare Application ____ 2-10

Building an AccessWare Graphical User Interface

Basic Concepts ____ 3-2
Guidelines for Creating Interfaces ____ 3-2
Suitability ____ 3-2
Consistency ____ 3-3
Presentation ____ 3-3
Overview to Creating a User Interface ____ 3-5
Resource Files ____ 3-5
Window Items ____ 3-5
The Item Attributes ____ 3-6
Application Actions ____ 3-7
Built-in Actions ____ 3-7
The Script Interface ____ 3-7
The Programmatic Interface ____ 3-7
The ACTION Attribute ____ 3-7
Other Features of AccessWare ____ 3-8
Use of Icons ____ 3-8
Color Palettes ____ 3-8
Help, Prompts, Tool Tips, and Messages ____ 3-9

Creating a Resource File 3-10
File Naming Conventions 3-10
Structure of a Resource File 3-10
Layout of the Resource File 3-11
Comment Lines 3-11
Item Ordering in a Resource File 3-12
Defining Items in the Resource File 3-12
The Item Definition 3-12
The Item QID Attribute 3-13
The Identifier of a Window 3-13
The QWINDOW Item 3-13
The Main Window 3-13
Attributes Which Characterize the Application 3-14
Defining Items in a Window 3-15
Some General Characteristics 3-15
Creating the Appearance 3-15
QPANEL Item 3-15
QLIST Item 3-16
Scrollbars and Headers 3-17
Hierarchical Trees 3-17
QBUTTON Item 3-18
QCHOICE Item 3-18
QTEXT Item 3-19
QMESSAGE Item 3-20
QMENU Item 3-21
QMENUITEM Item 3-21
QCONTROLSTATE and QEXPRESSION Items 3-21
QPOPUP Item 3-22
QGROUP Item 3-22
QTABCONTROL Item 3-23
QMENUBAR Item 3-23
QSEPARATOR Item 3-23
QSLIDER Item 3-23
Item Attributes 3-23
Global Variables in Attributes 3-24
Window Item ACTIONs 3-25
Invoking Actions from the Interface 3-25
Using the ACTION Attribute 3-26

Working with Lists 3-27
Defining the List Contents 3-27
List Headers 3-27
Scrolling of Lists 3-28
Scrollbars 3-28
Scroll Buttons 3-29
Scrolling from the Application 3-29
Data Selection from Lists 3-30
Popup Menus 3-30
Obtaining the Selected Row 3-30
Displaying Trees in AccessWare 3-32
Defining the List Contents File for a Tree 3-32
Use of Features of AccessWare Lists with Trees 3-33
Working with Menus 3-35
Ordinary Menus 3-35
Extended Menus 3-35
The Menu Item Definition File 3-35
The Menubar Definition File 3-37
Invoking a Menu Through the Interface 3-37
Dropdown and Pulldown Menus 3-37
Walking Menus 3-38
Popup Menus 3-39
Creating and Using Icons 3-40
Use of Icons in AccessWare 3-40
QWINDOW Items 3-40
QBUTTON Items 3-40
Creating an AccessWare Icon 3-41
Using Colors in AccessWare 3-43
Defining the Color Palette to Use 3-43
Creating a Color Palette File 3-43
Providing Context Sensitive Help 3-46
Defining a Prompt Action 3-46
Creating a Message Field 3-47

Setting Up a Help Window ____ 3-48
The HELP Window ____ 3-48
Toggling the Help Action ____ 3-49

AccessWare Item Definition Reference

Overview ____ 4-2

Window Item Reference ____ 4-3
QWINDOW Item ____ 4-4
Size and Position ____ 4-4
Definition of Items in a Window ____ 4-5
QPANEL Item ____ 4-5
QBUTTON Item ____ 4-6
QCHOICE Item ____ 4-7
QMESSAGE Item ____ 4-8
QTEXT Item ____ 4-8
QLIST Item ____ 4-9
QMENU Item ____ 4-10
QMENUITEM Item ____ 4-11
QCONTROLSTATE Item ____ 4-12
QEXPRESSION Item ____ 4-12
QPOPUP Item ____ 4-13
QGROUP Item ____ 4-13
QTABCONTROL Item ____ 4-14
QMENUBAR Item ____ 4-15
QSEPARATOR Item ____ 4-15
QSLIDER Item ____ 4-15
QATTREND Keyword ____ 4-15

Item Attribute Reference ____ 4-16
Item Attributes Summary ____ 4-16
Naming Items ____ 4-18
QID ____ 4-19

Attributes to Position Items ____ 4-19
XLOC, YLOC ____ 4-19
XPOS, YPOS ____ 4-20
XOFFSET, YOFFSET ____ 4-21
XAFTER, YAFTER ____ 4-22
Attributes to Define the Size of Items ____ 4-23
WIDTH, HEIGHT ____ 4-23
MINWIDTH, MINHEIGHT ____ 4-23
MAXWIDTH, MAXHEIGHT ____ 4-24
COLS, ROWS ____ 4-24
MAXROWS, MAXCOLS ____ 4-25
Attributes to Specify Behavior of an Item When Resized ____ 4-25
MOVEX ____ 4-25
MOVEY ____ 4-25
STRETCHX ____ 4-25
STRETCHY ____ 4-25
Attributes to Define the Visual Style of an Item ____ 4-25
ACTIVE ____ 4-25
VISIBLE ____ 4-26
ACTIVEDEP ____ 4-26
VISIBLEDEP ____ 4-27
DEPTH,INVERT ____ 4-27
ETCH ____ 4-28
STYLE ____ 4-29
LAYOUT ____ 4-30
ICON,DICON ____ 4-31
ICONLABEL ____ 4-32
Attributes to Control the Color of Items ____ 4-33
ITEMCOLOR,TEXTCOLOR ____ 4-33
BACKCOLOR,SELECTCOLOR ____ 4-33
GREYOUTCOL ____ 4-34
Attributes to Assign Labels to the Item ____ 4-35
LABEL ____ 4-35
Attributes to Define the Field Contents ____ 4-37
LISTFILE ____ 4-37
ITEMFILE ____ 4-38
LISTMENU ____ 4-38
MAXVALUE, MINVALUE ____ 4-39

Attribute to Supply Initial Defaults to an Item ____ 4-39
VALUE ____ 4-39
Attributes to Assign Special Characteristics ____ 4-41
STATE ____ 4-41
MODE ____ 4-43
DEFAULT ____ 4-44
FOCUS ____ 4-45
OVERVIEW ____ 4-45
INTERVAL ____ 4-46
Miscellaneous Attributes for Items ____ 4-46
FONT ____ 4-46
RULES ____ 4-47
UDATA ____ 4-47
Attributes to Invoke the Help Facilities ____ 4-48
PROMPT ____ 4-48
HELPFILE ____ 4-49
HELPWIN ____ 4-49
TIP ____ 4-50
Attributes That Affect the Whole Application ____ 4-50
RESIZE ____ 4-50
SPLASH ____ 4-51
NOBANNER ____ 4-51
NOSTRIPE ____ 4-51
PALETTE ____ 4-52
GLOBVARS ____ 4-52
ACTION Attribute Reference ____ 4-54
Invoking ACTIONs in AccessWare ____ 4-54
Form of the ACTION Attribute ____ 4-55
ACTION Attribute Summary ____ 4-55
Application Interface Coding Actions ____ 4-56
The Action Command File ____ 4-57
CF — Action Command File ____ 4-57
ACTIONS That Operate on Windows ____ 4-58
CL — Close Window ____ 4-58
EX — Exit Window ____ 4-58

Actions That Modify the State of an Item _____ 4-59
GO — Grey Out Item _____ 4-59
GI — Grey In Item _____ 4-59
HI — Hide Item _____ 4-60
SH — Show Item _____ 4-60
Actions That Set Values for Items _____ 4-60
SV — Set Value _____ 4-60
Actions That Denote Special Fields _____ 4-61
ME — Message Action _____ 4-62
PR — Prompt Action _____ 4-62
Actions That Invoke Menus and Popups _____ 4-63
MU — Invoke a Menu _____ 4-63
POP — Invoke a Popup Window _____ 4-63
POPREL — Invoke and Position a Popup Window _____ 4-64
Actions to Call Scripts _____ 4-64
SY — System Call _____ 4-64
SCR — Run Script _____ 4-65
UL — Update List _____ 4-65
MACRO — Define a Macro _____ 4-66
Actions That Issue CADDS Commands _____ 4-66
CAD — Send CADDS Command _____ 4-66
CAD1 — Send CADDS Command without CR _____ 4-67
Special Actions to Enable Scrolling of QLISTs _____ 4-67
HEAD — Header Bar Action _____ 4-67
HS — Horizontal Scrollbar _____ 4-67
VS — Vertical Scrollbar _____ 4-68
Scrolling Actions for QBUTTONs _____ 4-68
Actions to Toggle the Help Facility _____ 4-68
HELP — Toggle Help Action _____ 4-68

AccessWare Script Interface

Overview ____ 5-2

Using the Script Interface ____ 5-3

Displaying Messages from a Script ____ 5-3

Script Interface Examples ____ 5-4

Passing Arguments to a Script File ____ 5-4

Executing the Script File ____ 5-5

Updating a Text Field ____ 5-6

Description of the ACTION Attribute ____ 5-8

Updating List Contents ____ 5-8

Description of the ACTION Attribute ____ 5-10

AccessWare Programmatic Interface

Overview ____ 6-2

Applying an ACTION ____ 6-3

Using Libraries ____ 6-3

Structure of the Application Code ____ 6-4

The main Procedure ____ 6-4

The aw_user_exit Procedure ____ 6-4

Supplied include Files ____ 6-6

Sample aw_user_exit Procedure ____ 6-7

Modularising the User Exit Code ____ 6-8

Compiling and Linking Your Program ____ 6-10

Compiling ____ 6-10

Linking ____ 6-10

The AccessWare Procedure Library ____ 6-11

Item Referencing ____ 6-11

Summary of Procedure Calls ____ 6-12

Initiating and Exiting AccessWare ____ 6-13

Retrieving Item Values and Attributes ____ 6-13

Setting Item Values and Attributes ____ 6-15

Updating Item Appearance ____ 6-17

Procedures Associated with Lists ____ 6-17
Procedures Associated with Tree Nodes ____ 6-18
Miscellaneous Procedures ____ 6-18
AccessWare Programming Examples ____ 6-20
Passing Arguments to a Program ____ 6-20
Updating a Text Field ____ 6-23
Updating List Contents ____ 6-25
Creating a Confirmation Panel ____ 6-28
Scrolling to the Matching Line ____ 6-31

User Interface Examples

Example of Text Fields and Buttons ____ 7-2
The Window ____ 7-2
The Header ____ 7-2
The Footer ____ 7-3
Window Fields and Buttons ____ 7-4
The Complete Resource File ____ 7-5
Example of Lists and Choice Items ____ 7-6
LIST Items ____ 7-7
The Input Field ____ 7-8
QCHOICE Item ____ 7-8
The Complete Resource File ____ 7-9
References to Other Windows ____ 7-10
Example of a Menu Driven Interface ____ 7-11
Definition of the Main Window ____ 7-11
The Complete Resource File ____ 7-12
Defining a Menu Bar ____ 7-13
Creating Sub-menu Options ____ 7-14
Menu Item Definition Files ____ 7-17
The Complete Resource File ____ 7-18
Example of Extended Menus ____ 7-20
Menubar Definition File ____ 7-20
Menu Item Definition File ____ 7-20

Attribute Cross-Reference Table

Attribute Cross-Reference Table_____ A-2

Function Cross-Reference Table

Function Cross-Reference Table_____ B-2

Glossary

Preface

The *AccessWare User Guide* explains the installation procedure and how to build a user interface with AccessWare software. It also explains the use of Script Interface to execute external scripts and the Programmatic Interface to link to application programs.

Related Documents

The following documents may be helpful as you use *AccessWare User Guide*:

- *AccessWare Function Reference*
- *AccessWare Quick Reference*
- *Customizing EPD.Connect*

Book Conventions

The following table illustrates and explains conventions used in writing about Optegra applications.

Convention	Example	Explanation
EPD_HOME	`cd` $EPD_HOME`/install` (UNIX) `cd` %EPD_HOME%`\install` (Windows)	Represents the default path where the current version of the product is installed.
Menu selections	Vault > Check Out > Lock	Indicates a command that you can choose from a menu.
Command buttons and options	Mandatory check box, Add button, Description text box	Names selectable items from dialog boxes: options, buttons, toggles, text boxes, and switches.

User input and code	`Wheel_Assy_details`
	
	`-xvf /dev/rst0`
	
	`Enter command>` **`plot_config`**
		Enter the text in a text box or on a command line.
		
		Where system output and user input are mixed, user input is in bold.

System output	`CT_struct.aename`	Indicates system responses.
Parameter and variable names	`tar -cvf /dev/rst0` filename	Supply an appropriate substitute for each parameter or variable; for example, replace filename with an actual file name.
Commands and keywords	The `ciaddobj` command creates an instance of a binder.	Shows command syntax.
Text string	`"SRFGROUPA"` or `'SRFGROUPA'`	Shows text strings. Enclose text strings with single or double quotation marks.
Integer	n	Supply an integer for *n*.
Real number	x	Supply a real number for *x*.
#	`# mkdir /cdrom`	Indicates the root (superuser) prompt on command lines.
%	`%` `rlogin` remote_system_name `-l root`	Indicates the C shell prompt on command lines.
$	`$` `rlogin` remote_system_name `-l root`	Indicates the Bourne shell prompt on command lines.
>	`> copy` filename	Indicates the MS-DOS prompt on command lines.
Keystrokes	`Return` or `Control-g`	Indicates the keys to press on a keyboard.

Online User Documentation

Online documentation for each Optegra book is provided in HTML if the documentation CD-ROM is installed. You can view the online documentation from an HTML browser or from the HELP command.

You can also view the online documentation directly from the CD-ROM without installing it.

From an HTML Browser:

1. Navigate to the directory where the documents are installed. For example,

   $EPD_HOME`/data/html/htmldoc/` (UNIX)

   %EPD_HOME%`\data\html\htmldoc\` (Windows NT)
2. Click `mainmenu.html`. A list of available Optegra documentation appears.
3. Click the book title you want to view.

From the HELP Command:

To view the online documentation for your specific application, click HELP. (Consult the documentation specific to your application for more information.)

From the Documentation CD-ROM:

1. Mount the documentation CD-ROM.
2. Point your browser to:

   CDROM_mount_point`/htmldoc/mainmenu.html` (UNIX)

   CDROM_Drive:`\htmldoc\mainmenu.html` (Windows NT)

Printing Documentation

A PDF (Portable Document Format) file is included on the CD-ROM for each online book. See the first page of each online book for the document number referenced in the PDF file name. Check with your system administrator if you need more information.

You must have Acrobat Reader installed to view and print PDF files.

The default documentation directories are:

- $EPD_HOME`/data/html/pdf/`doc_number`.pdf` (UNIX)
- %EPD_HOME%`\data\html\pdf\`doc_number`.pdf` (Windows NT)

Resources and Services

For resources and services to help you with PTC (Parametric Technology Corporation) software products, see the *PTC Customer Service Guide*. It includes instructions for using the World Wide Web or fax transmissions for customer support.

Documentation Comments

PTC welcomes your suggestions and comments. You can send feedback in the following ways:

- Send comments electronically to `doc-webhelp@ptc.com`.
- Fill out and mail the PTC Documentation Survey located in the *PTC Customer Service Guide.*

Chapter 1

Introduction to AccessWare

This chapter introduces AccessWare and some of the basic concepts it uses to build Graphical User Interfaces.

- Introduction
- Basic Concepts of AccessWare
- Sample Resource File

Introduction

One of the most important aspects of any application program is the way the information is presented to the user. Many users nowadays expect to be presented with a modern mouse driven Graphical User Interface making extensive use of buttons, scrolling lists, icons, and menus.

Such interfaces require a great deal of development time using the resources of a skilled C programmer. Often, only the programmer has the ability to create and modify such a user interface, whereas an application support engineer or even the end user is often the best person to undertake this task.

The task of creating an interface requires much input from the eventual users of the system. This prototype stage often requires many changes, resulting in many iterations for recompiling the C code and relinking the application.

Once the interface has been designed, it must often run over several hardware platforms, and thus be portable from one window system to another. This aspect of development necessitates further code development time.

Using AccessWare to Create a GUI

AccessWare enables you to produce Graphical User Interfaces (GUIs) in a quick and easy manner. The interfaces are defined using a simple text language that is easy to understand. An entire interface can be developed and tested without writing a single line of C code.

AccessWare enables anyone to define a Graphical User Interface by creating and editing simple text files. The only skill that you need is to be able to edit a standard text file.

The text files are read by the AccessWare program at runtime to display the windows, text fields, buttons, lists, and menus. This allows you to test the GUI that you have created. No coding experience is needed to define and test the user interface.

Prototyping is easy since changes to the interface can be made by editing the text file. The effects of those changes can be seen immediately without needing to recompile and relink the programs.

The user interface definition is entirely portable. Once the user interface is defined, it will run across several hardware platforms and window systems without further modification.

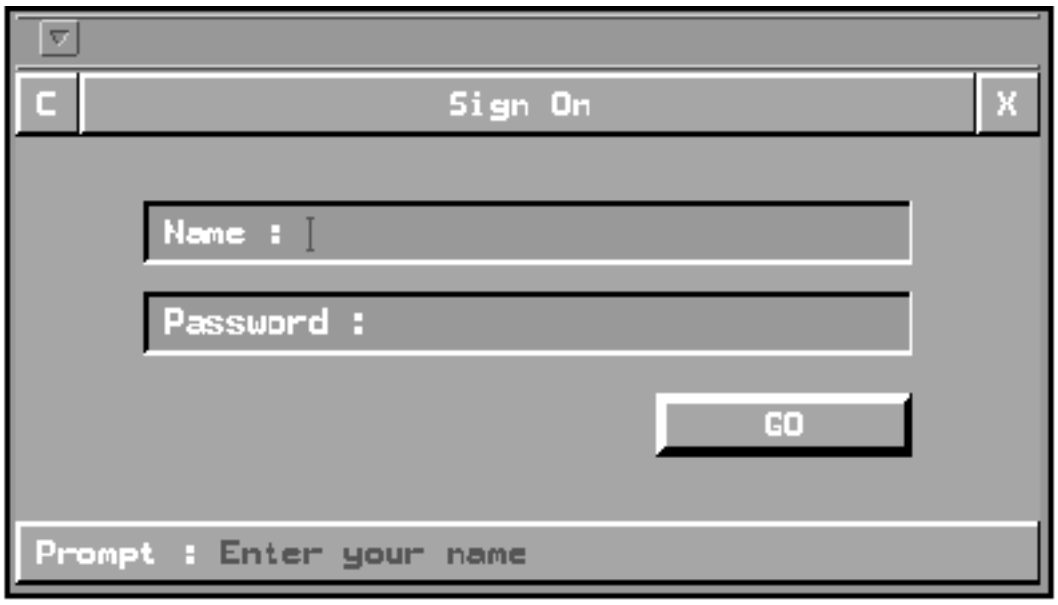
AccessWare can also link to a user's own C programs. A set of C procedures is provided, which can be incorporated into the user's own application to enable it to interact with the user interface. Thus, a skilled programmer can provide his/her application with a sophisticated Graphical User Interface.

Basic Concepts of AccessWare

A user interface in AccessWare is constructed from one or more individual windows. Each window can contain a number of data fields, lists, buttons, and menus. The user can enter data into text fields, select menu options, and click buttons. Any of these user interactions can result in predefined actions occurring within the interface or in the application.

A simple example of the type of window that can be produced in AccessWare is shown in the following figure.

Figure 1-1 Example of a simple Sign On Panel



This window is defined in one text file and is immediately displayed when the AccessWare program is run. If the text file is modified, the changes can be displayed immediately. This provides a very rapid prototyping and development environment.

Resource Files

In AccessWare each window is defined in a separate text file, called a resource file. This file contains details of the window size, its position, details of each menu, button, and field that appears within that window.

Within each resource file, every item (for example, menu, button, text field) of the GUI is defined by a list of attributes. These attributes define an item's position within the window, its size, visual appearance, special characteristics appropriate

to the item, and, if necessary, its purpose. For example, in the window shown in Figure 1-1, the text field defined to receive the user's password can have a 'secure' state attached, in order to inhibit echoing of the password as it is typed.

Application Actions

An action can be associated with any item in a window. This action forms the link between the user interface and the application code. In Figure 1-1, actions would be associated with the GO, C, and X buttons. Subsequent windows may include actions to read files. For example, these actions may take the filename, selected from a scrolling list, and then transfer the file from a database to the local disk for the user to view.

Actions from the AccessWare interface can be performed through an operating system script, a PC Batch file, or a C program using the Programmatic Interface.

AccessWare provides a set of standard actions to minimize the amount of coding and script writing. These include actions to hide and show items, redisplay lists, display other windows, and exit from windows.

Color, Icons and Help

AccessWare provides full control of the color of windows and items in the user interface. A color palette can be defined to enable the color shades to be uniquely defined for an individual application.

Icons can be defined to provide graphical images for use with items in the interface (such as buttons) and to enhance the visual appearance of the user interface.

AccessWare provides facilities to enable help in the form of prompts and other messages to be included in a user interface. In addition, context sensitive help windows can be defined for each window that appears when a Help button is clicked.

Sample Resource File

The resource file required to define the simple window shown in Figure 1-1 follows. This window allows a user to log onto a system.

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,200,LABEL,Sign On,QEND
# ================================================
QBUTTON
XPOS,0,YPOS,0,WIDTH,25,HEIGHT,25,->
LABEL,C,PROMPT,Turn the window into an Icon,->
TIP,Iconize Window,ACTION,CL,QEND
QPANEL
XPOS,25,YPOS,0,WIDTH,350,HEIGHT,25,LABEL, Sign On,QEND
QBUTTON
XPOS,375,YPOS,0,WIDTH,25,HEIGHT,25,   ->
LABEL,X,PROMPT,Exit from this application,ACTION,EX,QEND
# ================================================
# User name input text field
QTEXT
QID,sgn_name,XPOS,50,YPOS,50,WIDTH,300,HEIGHT,25,INVERT,1, ->
LABEL,Name:,PROMPT,Enter your name,QEND
# User password input field
QTEXT
QID,sgn_password,XPOS,50,YPOS,85,WIDTH,300,HEIGHT,25,INVERT,1, ->
LABEL,Password:,STATE,SECURE,PROMPT,Enter your password,QEND
# Action Button
QBUTTON
XPOS,250,YPOS,125,WIDTH,100,HEIGHT,25,DEPTH,4, ->
LABEL,GO,PROMPT,Enter the main application,ACTION,1000,QEND
# ================================================
# Define the footer prompt area
QPANEL
XPOS,0,YPOS,175,WIDTH,400,HEIGHT,25,LABEL,Prompt:, ACTION,PR,QEND
# End of window definition
QATTREND
# ================================================
```

Please note: The following is relevant for examples throughout the book.

- The lines beginning with # are comment lines within the resource file.
- *All* attributes for each item must be specified on a *single line*.

Chapter 2

AccessWare Setup Procedures

This chapter describes the installation and setup procedures for AccessWare.

- Understanding AccessWare Installation
- Setting the Development Environment
- Setting Environment Variables
- Executing an AccessWare Application

Understanding AccessWare Installation

AccessWare is installed in the location pointed by $EPD_HOME, that is, the directory where EPD.Connect is installed. $EPD_HOME points to `/opt/epd/dm/v60/bin` directory.

AccessWare System Contents

The following directory structure exists under $EPD_HOME:

Directory	Description
`bin`	The location of the `accesstool` programs to run under your host windowing system.
`src`	Example C source code to produce an application.
`obj`	Example `makefiles` (for compilation and linking) for various operating and host window systems.
`include`	Header files to include in C application code and a supplied icon for minimizing a window to an icon.
`lib`	AccessWare libraries for use in linking your C application code.
`data/palettes`	Example color palette files, including the default named `access.pal`.
`rules`	Examples of rules files for use with lists.
`examples`	Sample applications.

Warning

This must be valid against standard release procedures.

Setting the Development Environment

The AccessWare directories must be included in your UNIX search path. If AccessWare is installed in directory $EPD_HOME, add the following line in your `.login` file after the initial path setup.

```
set path = ($path $EPD_HOME $EPD_HOME/bin)
```

Please note: This directory name is assumed for the rest of the chapter. Substitute your own directory name if it is different.

An environment variable named `ACCESSDIR` can be set to point to the AccessWare directory. Define this in your `.cshrc` file:

```
setenv ACCESSDIR $EPD_HOME
```

Having modified the `.login` and `.cshrc` files, it is advisable to log out and log in again to activate the environment.

You are advised to make the entire AccessWare directory structure read only for AccessWare development work.

Setting the Runtime Environment

AccessWare is supplied with a subset of directories and files. A runtime application may need to access the following:

- `bin/accesstool` if you are using the accesstool program
- `palettes/access.pal` if you are using the default palette
- `include/access.icon` if you are using the default AccessWare icon

Setting Environment Variables

Environment variables are used to set standard AccessWare defaults as well as defaults for your application.

The use of UNIX environment variables and AccessWare global variables is interchangeable. Refer to “GLOBVARS” on page 4-52 for more information on global variables. The variables described on the following pages can be set in the `GLOBVARS` file or as an environment variable. If the variable is set in both places, the value in the `GLOBVARS` file takes precedence.

Summary of the Environment Variables

The following is a summary of the environment variables used by AccessWare.

Table 2-1 Summary of AccessWare Environment Variables

Environment Variable	Description
`ACCESSDIR`	Sets the AccessWare directory
`AW_RESOURCE`	Supplies the name of resource file
`AW_ITEM_COLOR`	Sets the item foreground color
`AW_LABEL_COLOR`	Sets the text color of a label
`AW_SELECT_COLOR`	Sets the selected item text color
`AW_BACK_COLOR`	Sets the item background color
`AW_PROMPT_COLOR`	Sets the color of the prompt text
`AW_MESSAGE_COLOR`	Sets the color of the message text
`AW_TEXT_COLOR`	Sets the color of the text specified by the `QTEXT` item
`AW_HILITE_COLOR`	Sets the default color of the left and top edges of an item.
`AW_LOLITE_COLOR`	Sets the default color of the right and lower edges of an item.
`AW_SCROLL_FACTOR`	Sets the scale of scrolling for trees using the mouse
`AW_MESSAGE_FILE`	Defines the name of the message file
`AW_XICON_TYPE`	Supplies the name of the window icon
`AW_WINDOW_STATE`	Specifies that all windows can be minimized to an icon irrespective of the state of the parent window
`ACCESSTRACE`	Sets the AccessWare trace level
`CMOM_DOMAIN`	Sets the application domain
`CMOM_DISPLAY`	Specifies the machine that displays the user interface defined in your main resource file
`AW_QUERY_ON_LIST`	Enables browsing the contents of a list when a key is pressed

Table 2-1 Summary of AccessWare Environment Variables (Continued)

Environment Variable	Description
AW_QUERY_ON_TEXT	Enables browsing the contents of a list when a value is typed in a QTEXT widget

The UNIX command to set a variable is as follows:

```
setenv variable_name variable_value
```

For more information on environment variable ACCESSDIR see "Setting the Development Environment" on page 2-3.

AW_RESOURCE

The AW_RESOURCE environment variable simplifies the execution of the AccessWare resource file.

To invoke your main resource file from the command line enter the following:

```
accesstool resource_filename
```

where:

- accesstool is the binary that reads the resource files.
- resource_filename is the name of your main resource file.

Using the AW_RESOURCE environment variable, you can define the name of your resource file in your .cshrc file. In this case you do not need to specify the name of the resource file to run accesstool.

Therefore, to run the executable accesstool using a resource file named my_resource.res you would add the following to your .cshrc file:

```
setenv AW_RESOURCE my_resource.res
```

This enables all windows to access to the my_resource.res file. Now, to invoke the my_resource.res file from the command line enter the following:

```
accesstool
```

Please note: This environment variable overrides any argument given to the accesstool application.

Colors as Environment Variables

Various color attributes can also be set through environment variables. The following table lists the color attributes and their equivalent environment variables:

Table 2-2 Color Environmental Variables

Attribute Name	Environment Variable Name	Description
ITEMCOLOR	AW_ITEM_COLOR	Sets the item foreground color
TEXTCOLOR	AW_LABEL_COLOR	Sets the text color of a label
SELECTCOLOR	AW_SELECT_COLOR	Sets the selected item text color
BACKCOLOR	AW_BACK_COLOR	Sets the item background color

The variables must be set to a value from your palette file. For example:

```
setenv AW_LABEL_COLOR 5
```

Please note: The color of the text in the prompt and message windows is set by separate environment variables.

AW_PROMPT_COLOR

The color used for displaying the text in prompt panels can be set by an environment variable named AW_PROMPT_COLOR. This variable should be set to a value from your palette file. For example:

```
setenv AW_PROMPT_COLOR 5
```

The default value is 4.

AW_MESSAGE_COLOR

The color used for displaying the text in message panels can be set by an environment variable named AW_MESSAGE_COLOR. This variable should be set to a value from your palette file. For example,

```
setenv AW_MESSAGE_COLOR 6
```

The default value is 3.

AW_TEXT_COLOR

The AW_TEXT_COLOR environment variable enables you to specify the default color of text specified by the QTEXT item in your user interface. For example:

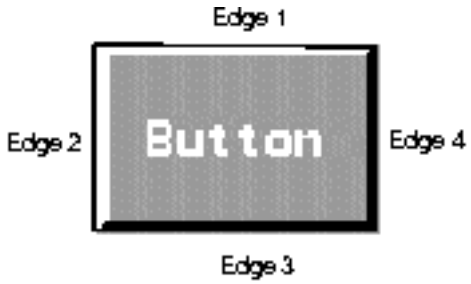
```
setenv AW_TEXT_COLOR 5
```

The default value is 4.

AW_HILITE_COLOR and AW_LOLITE_COLOR

The AW_HILITE_COLOR and AW_LOLITE_COLOR environment variables enable you to specify default color of the edges of an item to give it a three dimensional effect. The following figure illustrates the effect.

Figure 2-1 Button with AW_HILITE_COLOR and AW_LOLITE_COLOR defined



In the preceding figure, Edge 1 and Edge 2 are defined by the AW_HILITE_COLOR environment variable. Edge 3 and Edge 4 are defined by the AW_LOLITE_COLOR environment variable.

The default value for AW_HILITE_COLOR is 13, while the default value of AW_LOLITE_COLOR is 14. For example:

```
setenv AW_HILITE_COLOR 13
setenv AW_LOLITE_COLOR 14
```

AW_SCROLL_FACTOR

The AW_SCROLL_FACTOR environment variable enables you to set the scale of scrolling trees. Using the mouse you can drag a tree to view its different parts. For example:

```
setenv AW_SCROLL_FACTOR 1
```

The default value is 1.

AW_MESSAGE_FILE

This environment variable defines the pathname of the file that is used to pass messages to AccessWare. This variable is useful for applications using actions written in script files. Any message to be displayed to the user can be included in this file. For example:

```
setenv AW_MESSAGE_FILE /usr2/job/message_file
```

AccessWare checks the message_file following every action and displays its contents in the message area of the current window.

AW_XICON_TYPE

The format of an icon file defined for a QWINDOW item is defined by an environment variable AW_XICON_TYPE. For example:

```
setenv AW_XICON_TYPE aw
```

The variable takes one of the two values:

- xbm — specifies that the icon files are in native X format
- aw — specifies that the icon files are in AccessWare format

Please note: The AccessWare format icons can only be used when running under X windows.

AW_WINDOW_STATE

The AW_WINDOW_STATE variable enables you to minimize all windows in your user interface to an icon. The default value is 0. The value 1 enables popup windows to be minimized to an icon irrespective of the current state of the parent window. For example:

```
setenv AW_WINDOW_STATE 1
```

Please note: Only windows that contain RESIZE,4 as part of the QWINDOW definition in the resource file can be minimized to an icon when this environment variable is set.

ACCESSTRACE

The level of trace output to be displayed is set by the ACCESSTRACE environment variable. This variable should be set to a value from 0-15, with 0 as normal. The output traces calls to AccessWare actions and procedures. For example:

```
setenv ACCESSTRACE 5
```

CMOM_DOMAIN

The CMOM_DOMAIN environment variable enables you to receive CMOM messages by specifying an application domain. For example:

```
setenv CMOM_DOMAIN EPD.Connect
```

CMOM_DISPLAY

The CMOM_DISPLAY environment variable enables you to specify the machine that displays the user interface defined in your resource file. For example:

```
setenv CMOM_DISPLAY <machine_name>:0.0
```

AW_QUERY_ON_LIST and AW_QUERY_ON_TEXT

The AW_QUERY_ON_LIST and AW_QUERY_ON_TEXT environment variables enable you to use the combo box feature.

The default value for both these variables is 1. For example,

```
setenv AW_QUERY_ON_LIST 1
setenv AW_QUERY_ON_TEXT 1
```

Executing an AccessWare Application

To execute and test an interface developed with AccessWare use the following command:

`accesstool` resource_file_name

The `accesstool` binaries are available for a variety of hardware platforms and host window systems. The `accesstool` program resides in the $EPD_HOME`/bin` directory.

Please note: When running AccessWare application, a warning is given if an environment variable used by AccessWare is not set.

Chapter 3

Building an AccessWare Graphical User Interface

This chapter provides basic guidelines for building a user interface with AccessWare.

- Basic Concepts
- Overview to Creating a User Interface
- Creating a Resource File
- Defining Items in a Window
- Working with Lists
- Displaying Trees in AccessWare
- Working with Menus
- Creating and Using Icons
- Using Colors in AccessWare
- Providing Context Sensitive Help

Basic Concepts

Many applications use Graphical User Interfaces which comprise a number of windows. These windows may be displayed individually or many windows may be displayed simultaneously.

A window may include any or all of the following:

- Text fields that allow you to enter data.
- Scrolling lists to display lists of information from which one or more items can be selected.
- Buttons to select required options, which initiate actions within the application.
- Menus to select an option from a list of options.
- Panels for organizing and presenting the items.

AccessWare enables you to build Graphical User Interfaces that include any or all of the above. Users interacting with these items may initiate actions within the interface or within the application.

AccessWare also provides the environment within which windows that make up the user interface can be defined and tested.

Guidelines for Creating Interfaces

There are no hard and fast rules for creating user interfaces. Each application is different and has different considerations. However, some guidelines apply to all interfaces. They are:

- Suitability
- Consistency
- Presentation

When creating a new interface, first define the rules so that all aspects of the interface are handled consistently.

Suitability

A user interface must be appropriate to and suitable for the application.

Provide a sensible hierarchy of windows and menus: Operations that are related or performed at the same time should be accessed via the same panels

and/or menus. Having to make a number of selections to reach a frequently used item is also frustrating.

Provide a logical structure: The order in which the user is likely to perform operations should be reflected in the layout and presentation of the options in the interface.

Consistency

A user interface must be consistent across all its windows. The same style and presentation should be used throughout. Consider the following while creating your interface:

Text fonts and case:

- Keep text fonts consistent. If you use a different font, ensure the reason is obvious to the user.
- Keep the text case consistent. Remember that words in uppercase stand out, so use upper case sparingly and for a good reason.
- Labels with initial capital letters are easier to read, look better, and occupy less space.
- Present all messages in a similar style and grammar.

Overall appearance:

- Keep the size and spacing of text fields and buttons consistent.
- Ensure that the overall appearance of the interface is consistent.
- Always use the same header style at the top and the same footer layout at the bottom.

Presentation

The interface should be pleasing in appearance. It is easier to use and less strain for the user if it is interesting and attractive.

Layout: The windows should be laid out carefully. Use plenty of white space and do not overfill a window. Limit the buttons, boxes, and other information in a window to those a user can easily manage. The main windows can offer more options, but sub-windows and popups in particular should be spartan.

Use of graphics: Graphic images are always more interesting than text and are often easier to recognize. Graphic icons are smaller than text on buttons, and

therefore more suitable where a large group of buttons must be shown in a limited space.

Color is important: Avoid too many colors. Bright multicolored interfaces are initially eye catching, but can prove tiring during use. Neutral, cool colors (blues and greens) are more restful and are excellent for backgrounds. Warm colors (reds, yellows, and oranges) are more prominent and better suited for text and important buttons. Use different colors to distinguish between different types of text and ensure that text shows up clearly against its background.

Overview to Creating a User Interface

Building a user interface with AccessWare involves defining a number of windows, with each window described in a separate file called a resource file. Within each resource file, you can define items to be displayed in that window.

Items may be given an `ACTION`, which is initiated when it is selected. The action can be a built-in AccessWare `ACTION` or it can run a separate script or program.

Icons can be defined to include graphical representations of items in the interface, and a customized color palette can be used to further tailor the interface to your requirements.

Resource Files

Items of a user interface are defined in resource files using a text editor. The language used in these resource files involves English-like keywords that define the items and give them specific characteristics, such as their position, size, appearance and function.

Thus, each resource file defines the following:

- The window and its attributes
- Items to be displayed in that window and their attributes
- References to other files that form part of the interface

Items in the resource file are introduced by a keyword that defines the item. This is followed by a list of attributes for that item that describe its physical and logical characteristics, and the actions that it invokes.

Window Items

The following table describes the item types are used in AccessWare user interfaces:

Table 3-1 Description of AccessWare Items

Item	Description
`QWINDOW`	Defines a new window. This is the first item defined in the resource file.
`QPANEL`	Defines a rectangular area within a window and is used for visual effect.
`QBUTTON`	Defines rectangular buttons within a window. You can select buttons to set options or to invoke an immediate action.

Table 3-1 Description of AccessWare Items

Item	Description
QCHOICE	Displays a set of options for both single and multiple selections.
QTEXT	Allows you to display and/or enter single line text field text.
QMESSAGE	Provides an area in the window for the display labels and text messages.
QLIST	Displays a scrollable list of information for the selection of items for an application.
QMENU	References the definition of a menu used in conjunction with other window items.
QPOPUP	Provides the link to the definition of a window in another resource file.
QMENUITEM	Defines a single menu item.
QCONTROLSTATE	Controls the state of QMENUITEM. The control state can be visibly dependent or active dependent.
QEXPRESSION	Defines the boolean of two control states.
QGROUP	Groups a set of items for common actions.
QTABCONTROL	Generates tab controls to divide a window into logical areas or to provide a title for an information panel.
QMENUBAR	Generates top level menus for an application.
QSEPARATOR	Generates a separation within a panel or window.
QSLIDER	Generates a slide bar to adjust a numeric value.

The Item Attributes

A set of attributes is applied to each item in a window to define its physical and logical characteristics. The window also has its own set of attributes. The attributes for an item include:

- an item QID — the name that uniquely identifies it
- the size and location of the item
- its visual appearance, styling and color
- how the information and/or data is displayed
- the actual field contents of an item
- special characteristics of particular items

The attributes for an item are listed on the line following the item keyword. An attribute is also introduced by a keyword. A reference to the full set of attributes available for characterizing AccessWare items is given in the “Item Attribute Reference” on page 4-16.

Application Actions

An ACTION can be associated with most items in a window. An action forms the link between the user interface and the application. Actions are associated with buttons and initiated when the button is clicked. However, actions may be linked to any item in the window. Actions may also perform operations associated with the interface, such as updating a displayed list.

AccessWare provides two standard ways of invoking actions:

- Through a recognized standard built-in AccessWare ACTION.
- Through a call to an external script or an application program.

Built-in Actions

AccessWare has some standard built-in ACTIONs to minimize the amount of coding and script writing. These perform various basic functions, for example display other windows, scroll lists, and hide or show items. Using these built-in actions, many applications can be written without the need for C programming.

The Script Interface

The Script Interface invokes a script file to undertake certain actions. This allows more sophisticated applications to be built, without the need for knowledge of C coding. Refer to Chapter 5, “AccessWare Script Interface” for more information.

The Programmatic Interface

The Programmatic Interface provides most flexibility. It includes a suite of procedure calls that access items and their contents in the interface, enabling freely defined actions or sets of actions to be individually constructed in any manner the programmer chooses. Refer to Chapter 6, “AccessWare Programmatic Interface” for more information.

The ACTION Attribute

An ACTION is associated with an AccessWare item by the ACTION attribute. This comprises the ACTION keyword followed by a further keyword that identifies the action, followed by arguments that are required to process the action, if appropriate. For more information on the ACTIONs that can be invoked by AccessWare items see “ACTION Attribute Reference” on page 4-54.

Other Features of AccessWare

AccessWare enables you to customize a user interface and make it visually appealing, user friendly and easy to use. It allows:

- the use of icons
- color palettes, and
- help files, prompts, tool tips, and messages.

User prompts and Help features are an essential part of any user interface, both for guiding and assisting the user in the execution of tasks and for communicating warnings and errors.

Icons and color are very important in tailoring a user interface. They enable graphic images to be associated with the command actions. Customer logos and colors that are appropriate to a company's standards can also be created.

Tool tips provide quick information about actions that would occur when the user clicks a button.

Use of Icons

Icons are used in interfaces to provide a graphical representation of an item or an option. In AccessWare they can be used on buttons, in lists, and to represent the window when it is closed.

AccessWare icons are defined in a text file in the AccessWare icon file format. The setting of each pixel in the icon is defined by a hexadecimal number corresponding to a color in the palette file. For more information see "Creating and Using Icons" on page 3-40.

Color Palettes

AccessWare enables the color of windows and items in the user interface to be defined to suit the user's own requirements.

This is achieved using a color palette file containing a table of colors. Each color is given a color number and defined in terms of the amount of red, green, and blue. Sixteen different colors may be used within a single user interface, selected from a total range of 16 million shades.

The color numbers are referenced by certain item attributes to apply the required shade to an item. For more information on control of color see “Using Colors in AccessWare” on page 3-43.

Help, Prompts, Tool Tips, and Messages

AccessWare provides help in the form of prompts, messages, tool tips and the context sensitive online help files of the user interface.

A text field can be defined as a prompt field, to display prompts as the user moves the mouse cursor into other fields or over buttons. Context-sensitive Help windows can be provided, and displayed on selecting an item when the Help action has been toggled on. The content of the Help window is a scrollable area of text information read from a standard text file.

A message area can also be defined in a window so that warnings, errors, or any other information can be displayed to the user.

You can access the information provided by a tool tip by moving the mouse cursor over that button.

Creating a Resource File

Each AccessWare window and menu is defined in a separate text file referred to as a resource file. This file contains the definition of the window, its position, size and other attributes, followed by the definition of all the fields, buttons, menus and, other items that appear in the window.

A resource file includes references to other resource files for further windows that form a part of the same interface. It may also include references to Menu Item Definition Files to invoke menu options.

File Naming Conventions

The only limits on the name of a resource file are those imposed by the operating system. However, a file extension recommended is `.res`, which then identifies the file as a resource file. For ease of recognition, use an English-like description to define its function.

For example,

```
sign_on.res: an initial sign on window.
```

Every window, field, and button, must have a unique name across the entire user interface. This makes every item on every window available for access by the application at any time, whether or not the window is displayed to the user.

Structure of a Resource File

A resource file comprises a number of sections:

- The first section defines the window, with any window specific parameters or attributes. This is followed by any items which can be regarded as window header items.
- The next section contains definitions of the fields and buttons or other items contained within the main body of the window.
- The third section defines any items which can be regarded as window footer items. For example, prompt and message fields.
- The final section refers to any external definitions relating to this window, for example, the menu content definition files.

Please note: The items that make up a window header or footer are just standard window items. They are separated from the main body of the file in order to make the resource file easier to read. It also enables you to be consistent in style

across the complete interface, which may comprise a large number of individual resource files.

Layout of the Resource File

The basic layout of a resource file is as follows:

```
QWINDOW
window_attribute_list QEND
item _type
item_attribute_list QEND
QATTREND
```

- QWINDOW introduces the definition of the window

  window_attribute_list is a list of attributes to define the window.

- item_type within a QWINDOW item can be any of the following: QPANEL QBUTTON QCHOICE QTEXT QMESSAGE QLIST QMENU QPOPUP QMENUITEM QCONTROLSTATE QEXPRESSION QGROUP QTABCONTROL QSEPERATOR QSLIDER QMENUBAR.

Please note: You can repeat an item_type within a QWINDOW.

- item_attribute_list is a list of attributes to define the item.

  An attribute is defined by a keyword followed by the arguments. All the attributes must be given on a single line in the resource file. This must be the line immediately following the item type. Blank or comment lines are not allowed within the item definition. Attribute keywords and arguments are all separated by commas.

- QEND terminates the list of attributes for each item definition.
- QATTREND terminates the file.

Please note: An item definition is effectively delimited by the item keyword at the start and a QEND at the end. For example, QBUTTON begins the definition of a button and QEND completes the definition.

Comment Lines

Comment lines can be added to the resource files, and are denoted by the # symbol in the first column of any row. They must appear either before or after item definitions.

Item Ordering in a Resource File

The order of items in the resource file should be as follows:

1. `QWINDOW`

   This is the first non-blank, non-comment line in the file.

2. Other main items: `QBUTTON`, `QPANEL`, `QCHOICE`, `QTEXT`, `QMESSAGE`, `QLIST`, `QMENUITEM`,`QCONTROLSTATE`,`QEXPRESSION`,`QGROUP`, `QSLIDER`, `QSEPERATOR`,`QMENUBAR`

   These can appear in any order with their associated attributes.

Please note: You can specify a browse sequence for any text input fields. When the user uses the `TAB` key, the text cursor visits the text input fields in the order specified in the browse sequence.

3. `QMENU`

   Define any menu items following the main item definitions. This is necessary for AccessWare performance.

4. `QPOPUP`

   Define popup items referencing window definitions, in order to optimize AccessWare performance.

5. `QATTREND`

   This keyword terminates the resource file. It appears after all definitions are complete.

Please note: Careful structuring of the resource file makes it easier to create and maintain a user interface.

Defining Items in the Resource File

This section briefly outlines necessary considerations while creating AccessWare items within a resource file. For more information on AccessWare items see Chapter 4, “AccessWare Item Definition Reference”.

The Item Definition

All items are defined using their item keyword on one line, followed by a list of their attributes on the next line and terminated by `QEND`. For example,

```
QPANEL
QID,top_bar,XPOS,15,YPOS,15,WIDTH200,HEIGHT50,ETCH,2,QEND
```

The attributes define an item's name, its size, location, and visual effect. An item can also be given attributes that control the way it is used, the way it displays information, the way the user interacts with it, and the actions that it can invoke.

The Item QID Attribute

Every item in a window can have a name associated with it. This is defined by the `QID` attribute.

This name is used in the application code to identify the item in order to set item values, get user input, and make items invisible. It could appear as the argument of an attribute of another item to reference this item, or as an argument in a procedure call. An item does not have to be displayed to be referenced.

The Identifier of a Window

A `QWINDOW` item is referenced by the `QID` of the `QPOPUP` item which references its resource file. The main window, which does not have an associated `QPOPUP`, is referenced by the special identifier `mainwin`. Menus are referenced by the `QID` of the `QMENU` item which references their Menu Item Definition File.

Please note: The `QID` names must be unique across all items in ALL windows in an application. For example, use an item naming convention and prefix the name of each item with its window ID.

The QWINDOW Item

The `QWINDOW` keyword introduces the window definition. It is the first non-blank, non-comment line in the file. It carries the attributes that define the window.

The Main Window

The main window is the window displayed when an interface is initiated. It has the standard attributes for a `QWINDOW` item. It also has attributes to define the color palette, specify a global variables file, and specify the help window.

Please note: The main window may not be the principal window as far as the application is concerned, it could be just a sign on window that pops up the principal application window once a user has successfully logged on. It must however carry any main window attributes required for the application.

Each window in an interface is defined in terms of its size and position. Therefore, size and location attributes are mandatory. Other characteristics for an individual window and for the interface as a whole can be determined from defaults.

The XLOC and YLOC attributes are used to specify the location of the window. The origin for window positions is the top left corner of the screen. The location of the top left hand corner of the window is defined relative to the top left hand corner of the screen. The attributes WIDTH and HEIGHT are used to specify the size of the window.

Attributes Which Characterize the Application

Certain attributes are applied to the main window to reference files that set significant features for the application. These include:

- PALETTE: a customized color palette file
- HELPWIN: a resource file which defines the Help window
- GLOBVARS: a set of global variables for the interface

These attributes must be attached to the Main Window. They are referenced when the application is initialized.

Defining Items in a Window

This section provides an overview to defining items in an AccessWare window. It summarizes the main features of each item and describes their standard usage in an AccessWare interface.

In the resource file, the definition of the panels, buttons, text fields, and lists that appear in the window are introduced by their keyword, followed by a list of attributes to characterize them.

Some General Characteristics

Size and location attributes are mandatory for all AccessWare items, other than `QGROUP` and `QPOPUP` (the size of the popup window referenced by a `QPOPUP` item is defined in the window resource file).

The size and position of an item is defined in screen pixels. Some text fields can be more conveniently sized in terms of rows and columns of text.

The position of an item within a window is defined in pixels relative to the origin of the window, which is the top left corner of the usable window (inside the frame). Positions of items can also be defined relative to, or offset from, other items.

Please note: Define items that are physically related to each other sequentially. This allows you to easily modify the layout. These include labels to text fields, headers for lists, and sets of related buttons.

Creating the Appearance

There are several attributes that set the visual appearance of an item so that you can give it an appropriate styling for its function. Buttons can be shown raised, which is set using the `DEPTH` attribute. Text fields for user input can be recessed, which is achieved using both the `DEPTH` and `INVERT` attributes.

Such conventions are not always appropriate to an interface. Often, judicious use of the attributes, combined with panels, color, and etching to separate items, produces the desired appearance.

QPANEL Item

The `QPANEL` item defines a rectangular area within the window. It is used to divide a window into different physical areas. The panel can be given a visual appearance and a different color. There is no input associated with a panel.

Items can be nested, that is, placed one on top of another. Thus, a set of buttons or text fields can be superimposed on a panel.

Please note: The order of items in the resource file determines the order in which they appear on the display. Thus, to place a button on a panel, the `QPANEL` command must precede `QBUTTON` command.

A panel can be given a text label. The text appears in the top left corner of the panel. A panel may thus be used for labelling a field.

Example:

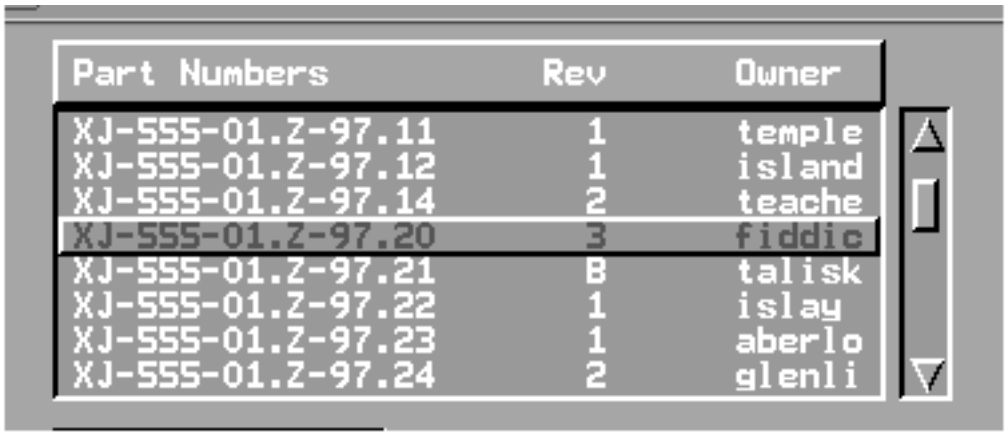
```
QPANEL
XPOS,25,YPOS,0,WIDTH,350,HEIGHT,25,LABEL,Sign On,QEND
```

A special use of panels is, in combination with a `QLIST` item, to provide a header and scrollbars for the list. The use of panels in this way is described later.

QLIST Item

A `QLIST` item displays a list of data to a user. Lists can be scrolled horizontally and vertically. Single or multiple items of data can be selected. A list can have a horizontal header that scrolls with the list contents.

Figure 3-1 Example of a `QLIST` with Scrollbar and Header



The contents of the list are read from a text file at runtime. A list is generated dynamically. Actions are available to update lists in conjunction with other operations initiated by the interface. For example, a list of files is updated when files are deleted.

Example:

```
QLIST
QID,dwg_list,XPOS,50,YPOS,50,WIDTH,300,HEIGHT,150, ->
ITEMFILE,/usr/lists/drawing_list,QEND
```

You can create a listbox that can take input from the keyboard. On typing a character in the listbox, the list scrolls to the first line that begins with the character you entered.

You can also, optionally, create a link between the textbox and the listbox, enabling you to use the combo box function. You can create this link between `QLIST` and `QTEXT` using the keyword `UDATA` in the `QTEXT` definition.

Example:

```
QTEXT
QID,MyTextBox,XOFFSET,110,YOFFSET,0,WIDTH,290,MAXCOLS,90,HEIGHT,2
5,INVERT,1,DEPTH,1,STRETCHX,100,MOVEY,100,MOVEX,100,UDATA,query_l
ist,QEND
```

`query_list` is the `QID` of the `QLIST`.

On creating this link and specifying one or more characters in the textbox, the listbox is queried for a matching value. If a matching value is found, then the list scrolls to that particular line. If no mathching value is found, then scrolling does not occur in the listbox.

Please note: This type of scrolling is unavailable for read-only textboxes.

Scrollbars and Headers

List headers and scrollbars are defined as separate window items. A list header is defined as a `QPANEL` item. An `ACTION` attribute associated with the `QPANEL` identifies it as a header bar. A list scrollbar is a `QPANEL` displayed with a movable bar and buttons at each end. An `ACTION` attribute to the `QPANEL` item identifies it as a scrollbar for the list and performs the scrolling function.

For information on creating `QLIST` items together with their scrollbars see “Working with Lists” on page 3-27.

Hierarchical Trees

A tree is a list of objects displayed hierarchically. A `STATE`, `TREE` attribute can be included in the attribute list to specify that the list contents file describes the structure of a tree and defines parent- child relationships. This structure is then

displayed as a tree. For more information see “Displaying Trees in AccessWare” on page 3-32.

QBUTTON Item

A QBUTTON item defines a single button. The button is used to invoke an action in the application. When you click the button, the button appears pressed.

Figure 3-2 Example of a Button and a Selected Button

The label on the button can be text or an icon. The colors of the button and its label can be changed for the pressed state. Two icons can be specified for a button, one for when it is up, the other for when it is pressed.

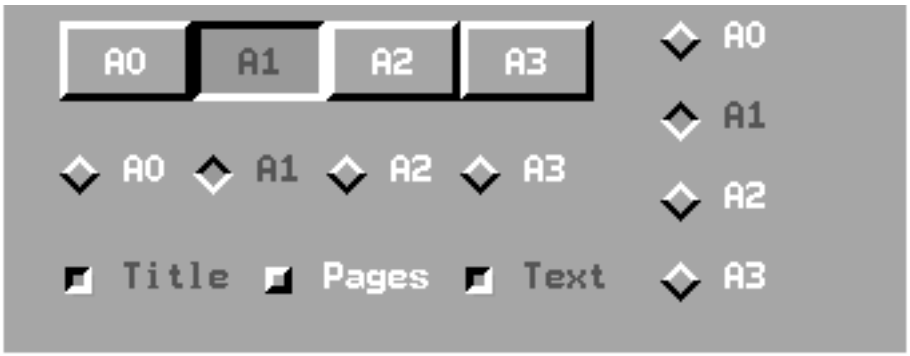
Example:

```
QBUTTON
XPOS,300,YPOS,200,WIDTH,100,HEIGHT,50,LABEL,GO, ->
PROMPT,Enter the main application,ACTION,1000,QEND
```

QCHOICE Item

A QCHOICE item displays a list of options from which the user can make one or more selections. It is used to enable a user to choose from a set of alternatives or select a set of options to be included when processing a command. A QCHOICE item is displayed as a set of buttons using a style dependent on the attributes given to it. Choice buttons in various styles can be created using AccessWare.

Figure 3-3 Example showing various styles of QCHOICE items



The LABEL attribute provides the list of options. Each option is separated by the character tilda (~). The option list is terminated by the keyword QQ. For example,

```
QCHOICE
QID,dwg_size,XPOS,100,YPOS, 200,WIDTH,40,HEIGHT,25, ->
LABEL,A0~A1~A2~A3~QQ,VALUE,2,QEND
```

Single or multiple option selection is specified by the MODE attribute. In single mode selection, a new choice deselects the previously selected option.

A default option is set by the VALUE attribute. It takes an integer representing the position of the default option. In the preceding example, VALUE,2 sets the default for the item dwg_size to A2.

A QCHOICE item can be displayed as toggle buttons rather than standard buttons (the default). The attribute MODE,TOGGLE displays the item as a set of toggle buttons with adjacent labels. The style of the buttons reflects the Motif style. If single option selection is specified, then diamond shaped buttons are shown; if MODE,MULTI is specified, then square buttons are displayed.

Please note: The size attributes specify the size of a single button. The space occupied by a QCHOICE item depends on the number of buttons and, if the STYLE,TOGGLE attribute is used, the length of the labels.

A full description of the QCHOICE attributes is given in Chapter 4, “AccessWare Item Definition Reference”.

QTEXT Item

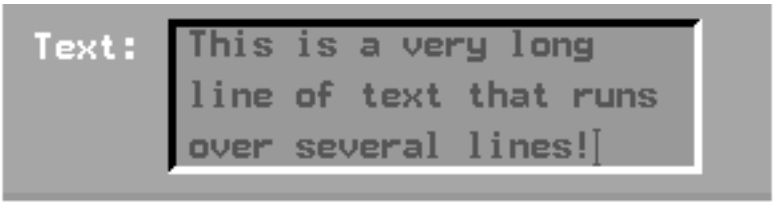
A QTEXT item accepts typed input and can be used as a data entry field. The text is echoed and in-line editing of the text string is allowed. Any printable characters are considered valid input.

You can perform cut, copy, and paste operations in the QTEXT field. You can cut or copy the value from a widget and paste it in the QTEXT widget and vice versa. You can use both the mouse and the keyboard to carry out cut, copy, and paste operations. You can also perform drag-and-drop operations in the QTEXT field.

Please note: You cannot perform cut, copy, and paste operations in the read-only textboxes.

The text field can be defined as a number of rows. Text intelligently wraps from one row to the next breaking at complete words and thus the total number of characters that can be entered may be less than that implied by the size attributes.

Figure 3-4 Example of a QTEXT Item for multiline text



The following example defines fields for entry of a user's name and password.

Please note: The use of an environment variable to supply a default username and a secure text field to enter the password.

```
QTEXT
QID,sgn_username,XPOS,10,YPOS,100,WIDTH,350,HEIGHT,25, ->
LABEL,Username:,VALUE,$USERNAME,INVERT,1,QEND
QTEXT
QID,sgn_password,XPOS,10,YPOS,150,WIDTH,350,HEIGHT,25, ->
LABEL,Password:,STATE,SECURE,INVERT,1,QEND
```

The order of the text fields in the resource file determines the order in which they are entered when the user presses the TAB key.

The MODE attribute can be used to provide some control over data entry, but data validation can only be done by the application.

Please note: For information on creating a link between a textbox and a listbox, see "QLIST Item" on page 3-16.

QMESSAGE Item

The QMESSAGE item is a simple field displaying a single line of text on the interface. No input is accepted in this field and the cursor skips this field when a user presses the TAB key around the input fields.

The QMESSAGE item can be used to display a simple message or a label against another field. For example,

```
QMESSAGE
XPOS,100,YPOS,200,WIDTH,100,HEIGHT,25,LABEL,Help,QEND
```

The QMESSAGE item can also take an ACTION attribute. Thus, you can use it to create an item in the window that invokes a menu or popup window.

QMENU Item

A QMENU item references a Menu Item Definition File. The menu supplies a list of items from which an option can be selected. An action is associated with each item. The QMENU item is referenced by another item in the window that invokes the menu.

The ITEMFILE attribute to the QMENU item specifies the name of the Menu Item Definition File, a text file holding the menu contents. The file is read at runtime and the menu sized as required to display the items. This enables the user to provide dynamic menus.

Please note: In the following example, the menu resides in the text file: /usr/menus/mymenu. However, the item that invokes the menu references menu_qid (the QID of the QMENU item). For example,

```
QMENU
QID,menu_qid,ITEMFILE,/usr/menus/mymenu,COLS,10,QEND
```

For more information on the structure of the menu text file and on setting up menus see “Working with Menus” on page 3-35.

QMENUITEM Item

The QMENUITEM item defines a single menu item of a menu such as FILE > NEW where FILE is the menu and NEW is the menu item.

If QMENUITEM entries are defined in a separate resource file called menus.r then there needs to be some reference for the QMENUITEM definitions in the application's main resource file. For this, the same procedure that is followed for popup windows resource files is used by having QPOPUP definitions for this file in the main resource file. This can be done as follows:

```
QPOPUP
QID,menuitems,STATE,DYNAMIC,VISIBLE,0,LABEL,->
$CA_RESOURCES/menus.r,QEND
```

Thus, the QID of QPOPUP item for menus.r resource file reference is used in the menus.r file for QWINDOW definition’s LABEL value.

QCONTROLSTATE and QEXPRESSION Items

The QCONTROLSTATE and QEXPRESSION items define the availability of menu items. These items can be used to make certain menu items inactive or invisible by controlling the state in which it displays.

The QCONTROLSTATE item is defined for each occurrence of ACTIVEDEP and VISIBLEDEP attributes in QMENUITEM definitions. The QEXPRESSION item is used to define the boolean of two or more control states.

If QCONTROLSTATE and QEXPRESSION entries are defined in a separate resource file called states.r then there should be some reference for the QCONTROLSTATE and QEXPRESSION definitions in the application's main resource file. For this, the same procedure that is followed for popup windows own resource files is used by having QPOPUP definitions for this file in the main resource file. This can be done as follows:

```
QPOPUP
QID,states,STATE,DYNAMIC,VISIBLE,0,LABEL,->
$CA_RESOURCES/states.r,QEND
```

Thus, the QID of QPOPUP item for states.r resource file reference is used in the states.r file for QWINDOW definition's LABEL values

QPOPUP Item

The QPOPUP item identifies another window that forms part of the same user interface. Other windows are defined in other resource files. Thus, the attributes to the QPOPUP item specify the name of the resource file containing the window definition. They also specify whether the window is displayed or hidden. The following example defines a popup window.

Example:

```
QPOPUP
QID,panel_2,VISIBLE,1,LABEL,/usr/apl_abc/panel_2.res,QEND
```

Please note: Items reference another window by using the QID of its QPOPUP item, which in turn identifies the window's resource file. Thus in the example, the resource file /usr/apl_abc/panel_2.res defines a window, but references to this window from elsewhere in the application interface are via panel_2 (the QID of the QPOPUP item).

QGROUP Item

The QGROUP item provides a way of grouping a set of items so that a common set of actions can be applied to them as though they were one item.

The items in the group are defined in a text file referenced by a ITEMFILE attribute to the QGROUP item. For example,

```
QGROUP
```

```
QID,group_1,ITEMFILE,/usr/apl_abc/item_groups/group_1,QEND
```

The file `/usr/apl_abc/item_groups/group_1` contains a list of items to be treated as one group. Refer to "QGROUP Item" on page 4-13 for more information.

QTABCONTROL Item

The `QTABCONTROL` item defines tab controls for a window. It is used for visual effects, for example, to divide a window into logical areas, or to provide a title for an information panel. For example,

```
QTABCONTROL
XPOS,10,YPOS,20,WIDTH,200,HEIGHT,20,LABEL,->
General~Attributes~QQ,VALUE,1,QEND
```

QMENUBAR Item

The `QMENUBAR` item defines top level menus for an application. For example,

```
QMENUBAR
XPOS,0,YPOS,0,WIDTH,100,HEIGHT,25,DEPTH,2,ITEMFILE,somefile,QEND
```

QSEPARATOR Item

The `QSEPARATOR` item generates a separation within a panel or window. For example,

```
QSEPARATOR
XPOS,10,YAFTER,2,WIDTH,150,HEIGHT,148,DEPTH,1,QEND
```

QSLIDER Item

The `QSLIDER` item generates a slide bar to adjust a numeric value. For example,

```
QSLIDER
QID,xsliderr,ETCH,4,XPOS,0,YAFTER,0,WIDTH,130,HEIGHT,25,->
LAYOUT,H,LABEL,0~100~QQ,VALUE,10,ACTION,1001,MINVALUE,0,->
MAXVALUE,100,VISIBLE,1,QEND
```

Item Attributes

Items in a user interface are characterized by their attributes. A considerable number of attributes may be given to an item in a window and the window can have a number of attributes.

The key points to note when entering item attributes are as follows:

- An attribute is defined by a keyword followed by its arguments.
- A comma terminates a keyword and each argument. Spaces and tabs are used only in text strings in the attribute list.
- All attributes must appear on a single line in the resource file.
- This line must immediately follow the item keyword. Blank lines and comment lines are not used within the definition of an item.
- The order in which attributes are defined is not significant.
- The definition of an item is completed by the keyword `QEND`.
- Attributes can take various types of arguments, including integer and real numbers, text strings, filenames, and recognized keywords.
- A text string embraces every character between the commas including all leading and trailing spaces.
- Filenames can be defined with full or relative pathnames.

Please note: Many attributes are common to most items. However, some are specific to individual items, as are some of the attribute values and keywords. In such cases, AccessWare does not always reject attributes that are inappropriate to a particular item, but ignores them.

The argument values specified in the resource file are the values that are assigned when the application initializes. Some of these values may be changed during execution of the application.

Chapter 4, “AccessWare Item Definition Reference” provides a reference to the complete set of attributes.

Global Variables in Attributes

A global variable can be used within the argument to an attribute. Global variables are used to specify values that are set outside the application to take effect for particular run.

They are used to provide data about the environment of the current user. They may be used to supply a directory name for certain files, such as data files which could vary from user to user and cannot be hard-coded into the interface.

They are also be used to provide directory names for certain files such as data listings, menus, and icons. This makes it easier to move an application from a development to a production environment and also enables the eventual users to further tailor an interface to their own environment.

Window Item ACTIONs

An `ACTION` can be associated with most items in a window. It forms the link between the user interface and the application. It is initiated as a result of some activity in the interface, such as the clicking of a button, and results in an event or action taking place in the application.

Although actions are associated with buttons they may be linked to any item in the window. Actions may also perform operations associated with the interface, such as updating a displayed list.

Invoking Actions from the Interface

AccessWare provides the following ways of invoking actions:

- Through a standard built-in AccessWare action, invoked by a recognized keyword.

  Standard actions are supplied with AccessWare to minimize the amount of coding and script writing. Some of these actions perform functions that would otherwise require a C program.

  One action is associated with one item. However, if a number of actions need to be executed, they can be entered into a command file which runs when the item is selected.

- Through a call to an external script using the Script Interface.

  The Script Interface runs a script file at the operating system level, providing access to items in the interface. It allows quite sophisticated applications to be built without C coding knowledge.

- Through the Programmatic Interface.

  The Programmatic Interface enables users to invoke their own programs from actions in an AccessWare interface. It includes a suite of C procedure calls that access items and their contents in the interface from within the user's own programs. Thus, window items can be manipulated in conjunction with events in the application. For example, buttons can be disabled when their actions are inappropriate.

The script action and Programmatic Interfaces are described in Chapter 5, "AccessWare Script Interface" and Chapter 6, "AccessWare Programmatic Interface."

Using the ACTION Attribute

If an action is associated with an AccessWare item, the attribute list must include the ACTION attribute.

The ACTION attribute can be specified for any item. For all items other than QWINDOW, the action is performed when the user clicks on the item concerned.

An ACTION specified for a QWINDOW item is performed when the window is initially created and again as the application exits. This provides a facility for initiating item values and for tidying up on exit.

The ACTION attribute includes a keyword that identifies the action. This can be followed by further arguments required for the action. For example, to run a command file the attribute list would include:

Example 1:

```
ACTION,CF,/usr/files/myaction_file
```

ACTIONs can also identify the way an item is used. For example, the ME action identifies an item that displays application messages.

Example 2:

```
ACTION,ME
```

An ACTION attribute may also be followed by an integer number which identifies a label in the application Programmatic Interface. The integer value must be greater than 500. Values less than 500 are reserved for internal use by the AccessWare application.

Example 3:

```
ACTION, 1000
```

A reference guide to the complete set of ACTION keywords that can be used in AccessWare interfaces is given in “ACTION Attribute Reference” on page 4-54.

Working with Lists

AccessWare can display lists of information to a user. The QLIST item type is used to define lists, any number of lists can be defined within a single window. Lists can be defined with the following characteristics:

- to scroll horizontally or vertically
- to scroll by means of scroll buttons or scrollbars
- to contain a list header that scrolls with the list contents
- to enable selection of single or multiple data items

A list can also be displayed as a tree, described in "Displaying Trees in AccessWare" on page 3-32.

When a list is defined, a set of attributes define the position and size of the list panel, the data that it contains, the way in which it is displayed, and how it behaves when the user interacts with it.

Defining the List Contents

The contents of a list are held in a text file. The ITEMFILE attribute defines the name of the text file. For example,

```
ITEMFILE,$LISTDIR/part_names
```

The argument to the ITEMFILE keyword is the filename of a text file containing data to be displayed. The contents of the file are displayed with the first row of the file at the top of the list.

If the file contents are updated, the displayed list is not updated until a request is made to re-display the list. This request may be initiated using either the AccessWare ACTIONs: SH or UL, or the C procedure call aw_update_list.

List Headers

It is often useful to include a line of column header information that does not disappear when the list is scrolled vertically but can be scrolled horizontally to maintain vertical alignment with the list data.

A list header bar is defined as a QPANEL. It is given an ACTION attribute called HEAD, which takes as its argument the QID of the QLIST item.

AccessWare searches the same directory as the list contents file for the header definition. It expects a text file of the same name as the data file but with an extension of `.hdg`. For example, `part_names.hdg`.

Example:

```
QLIST
QID,list1,XPOS,30,YPOS,30,COLS,35,HEIGHT,110,ITEMFILE,->
part_names,QEND
QPANEL
XOFFSET,0,YOFFSET,-25,COLS,35,HEIGHT,25,ACTION,HEAD,list1,QEND
```

AccessWare creates the `QLIST` and `QPANEL` items, writing the contents of the header file `. . ./part_names.hdg` to the panel. If the list is subsequently scrolled by n characters, the header also scrolls by n characters.

Scrolling of Lists

AccessWare can add scrollbars and define separate buttons that scroll a list either vertically or horizontally. Special AccessWare `ACTIONS`, attached to the items defined for this purpose, perform the scroll functions.

Scrollbars

A scrollbar enables a list to be scrolled. In an AccessWare interface it is defined as a `QPANEL` item and is displayed with a movable bar, the elevator, and buttons at each end.

Figure 3-5 A horizontal scrollbar

The elevator in a scrollbar indicates the portion of the list displayed in the `QLIST` window. The position of the elevator indicates the location in the file and the length of the elevator in proportion to the length of the scrollbar indicates how much of the file is displayed.

If the user clicks the scrollbar, the centre of the elevator is repositioned at the click point and the list is scrolled to display the corresponding section of the file. The list can also be scrolled by clicking on the buttons.

In the resource file, the scrollbars are defined as separate QPANEL items from the QLIST. They are defined by a VS or HS action (for vertical scrollbar or horizontal scrollbar), followed by the QID of the QLIST with which they are associated.

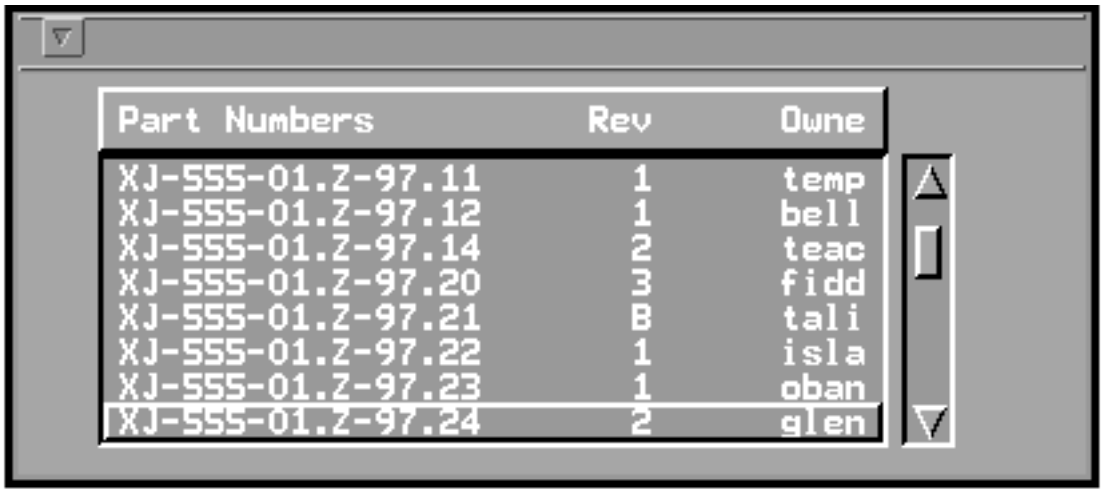
A scrollbar can be located anywhere within a window. When used to scroll a list it is positioned adjacent to the list. In the following example note the use of the XAFTER and YOFFSET attributes.

Example:

```
QPANEL
XAFTER,5,YOFFSET,0,WIDTH,2,HEIGHT,110,ACTION,VS,list1,VALUE,4,QEND
```

The number of rows (or characters in the case of a horizontal scrollbar) that a list is scrolled by is specified by the VALUE attribute in the definition of a QPANEL scrollbar.

Figure 3-6 Example of a QLIST with Vertical Scrollbar



Scroll Buttons

Individual buttons can be defined to scroll lists up, down, left, and right. They can be used instead of scrollbars or along with them. ACTION attributes attached to the QBUTTON items identify their scrolling function. A button used for scrolling may also be given a STYLE attribute, which displays the button as an arrowhead.

Scrolling from the Application

A list can also be scrolled from the application. The button scrolling ACTIONs can be used to scroll a list from an application written using scripts.

The C procedure calls `aw_scroll_to_line` and `aw_scroll_to_char` can be used to scroll the list to specified positions in the file.

Data Selection from Lists

The `MODE` attribute to a `QLIST` defines the selection mode for the list as either single item selection or multi item selection. The default is single item selection, that is, the currently selected row is deselected when a new row is selected.

To permit multiple selections of items from a list, the attribute `MODE,MULTI` must be included in the attribute list for the `QLIST` item.

Please note: For the multiple row selection mode, if you select a row that is already selected, it is deselected.

Popup Menus

Popup menus are displayed in the list panel when the right mouse button is clicked. A `LISTMENU` attribute to the `QLIST` is used to identify a popup menu associated with a list.

Example:

```
LISTMENU,/usr/menus/myfile_menu
```

These are used to offer a set of options for actions that are to be applied to selected rows of the listing.

Popup menus are further discussed in the section “Working with Menus” on page 3-35.

Obtaining the Selected Row

AccessWare enables the selected row or rows to be read by the application.

Single Item Selection Mode: In the Programmatic Interface the row number of the selected item in the list can be obtained from the argument to the `VALUE` attribute using the C procedure call `aw_get_item_value`. The row text is then obtained by invoking the procedure call `aw_get_list_line`.

Please note: In the Script Interface, the `VALUE` argument is passed as an argument with the `ACTION,SCR` action. Refer to Chapter 4, “AccessWare Item Definition Reference” for more information.

Multiple Item Selection Mode: In multiple selection mode, the selected row numbers can only be obtained from a C application. The procedure call `aw_is_item_bit_set` is used to interrogate each row number to check whether or not it is selected. The procedure call `aw_get_line` can then be used to get the row text for each selected row.

Displaying Trees in AccessWare

AccessWare provides the ability to display a list of objects that are hierarchically related as a tree. A tree listing is a special case of the `QLIST` item. The list contents file is expected to describe the parent-child relationships of objects to be displayed.

To display the contents file as a tree, the `QLIST` item must include the `STATE,TREE` attribute in its definition.

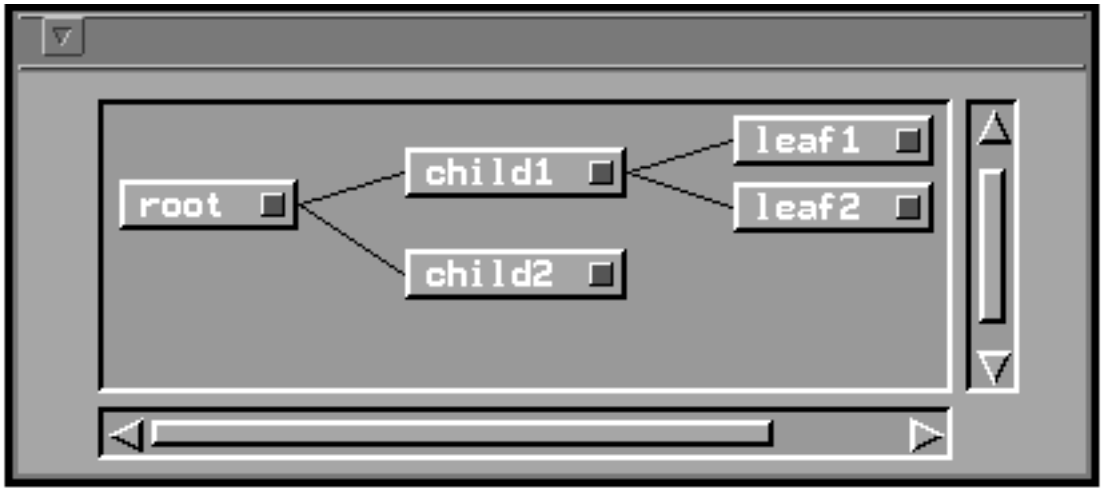
Defining the List Contents File for a Tree

The `ITEMFILE` attribute to the `QLIST` item supplies the name of the list contents file for a tree. This file contains a list of objects for display as nodes in a tree. For each node in the tree, the following must be specified:

- Item number: A sequential node reference number
- Node label: A name to identify the object node
- Parent no: The Item number of its parent node
- Class: Single character code
- Visibility flag: Set node visibility: 0/1 (1 = visible)
- Highlight: Highlight color from color palette
- Text color: The node text color
- Node access flag: Color of the node visibility flag
- Link line color: Color of the node link lines

A simple example of a tree containing a root node and four child nodes is shown in the following figure.

Figure 3-7 Example of a `QLIST` with a Simple Tree



The list contents file for the tree shown in the preceding figure is as follows:

```
1     root        0     R     1     0     11    1     4
2     child1      1     R     1     0     11    1     4
3     leaf1       2     R     1     0     11    1     4
4     leaf2       2     R     1     0     11    1     4
5     child2      1     R     1     0     11    1     4
```

Please note:

- Each in field in the list contents file must be specified on one line and separated by `TAB` spaces.
- The parent-child relationship is defined in the third column, this contains the node reference number of the parent.
- The Node access flag, shown as a square in the preceding example, serves two purposes:
  - The shape of the node indicates whether there are further nodes in the tree, but currently not displayed because they have been rendered invisible (the flag changes to a triangle to indicate that there are unseen nodes below this node).
  - The Nodes can be color coded to set some status associated with the application.
- The Class field is not currently used, but must included.

Use of Features of AccessWare Lists with Trees

The features of AccessWare lists can be used with trees in a similar way to text based lists. Horizontal and vertical scrollbars can be defined, they act upon the1fs

panel in which the list is displayed, scrolling the list to enable different areas of it to be viewed. A header panel can also be displayed.

A user can select nodes of the tree in the same way as selecting lines from the listing of a text file, by clicking the node. Popup menus can also be associated with a tree listing.

Working with Menus

Menus are a useful way of providing sets of related commands. They can be made available to users as dropdown or pulldown lists, invoked when the user selects an item in the window, or as a popup window from a mouse button. Menus are often context sensitive, that is, the options available depend on the current status of the application or items in the window. Menus can be created using either of the following methods, or a combination of both:

- Ordinary Menus (old format menus)
- Extended Menus

Ordinary Menus

AccessWare can create the following types of menus in a user interface:

- dropdown or pulldown menus
- walking menus
- popup menus

Menus in AccessWare must be defined in an external file, the Menu Item Definition File. They are introduced into an AccessWare user interface when a `QMENU` item is defined. This `QMENU` item is then related to the item which invokes the menu when the `ACTION,MU` attribute is defined.

Extended Menus

AccessWare also supports extended menus. Extended menus can be used to dynamically show, hide, activate, or deactivate menus. This means that the user can enable, disable, hide, or show any menu, at any time using the resource files and AccessWare APIs. Ordinary menus can be created as part of extended menus. The syntax for defining these menus is different from ordinary menus.

The Menu Item Definition File

The content of a menu is defined in an external text file. This text file defines the text to appear in the menu, the action to be performed when that item is selected, and sections within the menu. Using an external text file creates dynamic menus at runtime.

The format of a line in the Menu Item Definition File follows:

Menu_Name<TAB>AccessWare_Action
Menuitem_id<TAB>Menu_Name<TAB>EXTENDED

where

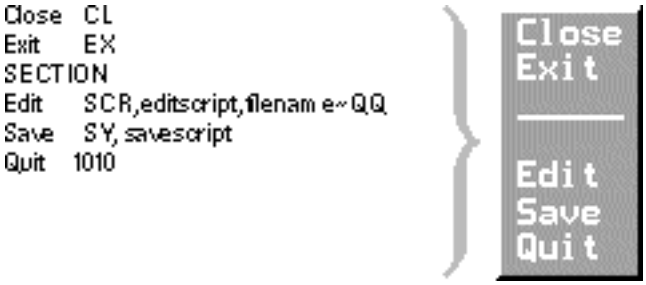
- Menu_Name refers to the name of the menu.
- AccessWare_Action refers to the action associated with that menu.
- Menuitem_id is the QID of QMENUITEM, which is defined in resource file.
- EXTENDED specifies that the menu item has extended capabilities.

Please note:

- It is mandatory to define the Menu_Name for EXTENDED menus. This overrides the LABEL attribute defined in resource file for QMENUITEM.
- All fields in the Menu Item Definition File are separated TAB spaces.

A line containing the keyword SECTION denotes a section break within the displayed menu, which is used to separate different types of options in the menu. The following figure is an example of a Menu Item Definition File and the corresponding menu it creates.

Figure 3-8 Example of a Menu Item Definition File



Please note: The actions specified in a Menu Item Definition File can include the standard built-in AccessWare actions and references to labels in the Programmatic Interface.

If no action is defined for the selected option in the Menu Item Definition File, AccessWare searches for any action defined against the QMENU item definition and initiates this action. This default action is performed only if no action is defined for the option in the file.

The Menubar Definition File

In order to understand the Menubar Definition File format, consider the following example:

```
&File<TAB>$CA_MENUS/ca_hfile.men
&Reports<TAB>$EPD_HOME/data/custmen/ca_men1.men
```

where, `$CA_MENUS/ca_hfile.men` and `$EPD_HOME/custmen/ca_men1.men` are text files (Menu Item Definition Files) defining menu items that can be ordinary menus or extended menus. All fields are separated by `TAB` spaces.

Invoking a Menu Through the Interface

A menu is introduced into an AccessWare user interface by defining the `QMENU` item. The menu is invoked by selecting another item which has an `ACTION,MU` attribute defined referencing the `QMENU` item.

Dropdown and Pulldown Menus

A dropdown or pulldown menu is a menu that drops to display a selection of choices when the user selects the menu header item.

Menus positioned along the top of a window are sometimes referred to as dropdown menus - the header item from which the menu is invoked contains a piece of text describing the menu contents. Similar menus, invoked from items placed anywhere in the window, are referred to as pulldown menus.

To create a dropdown or pulldown menu, define a `QMESSAGE` item as the menu header that invokes the menu when selected by the mouse. The `LABEL` attribute defines the piece of text which is displayed. It is common practice to place a horizontal line under the first character in the item label to denote its menu function.

Give the item an `ACTION` of type `MU`, with the `QID` of the `QMENU` item referencing the menu contents file as the argument. The menu name and menu file name are then associated with item.

Example:

```
QMESSAGE
XPOS,20,YPOS,30,WIDTH,150,HEIGHT,25,LABEL,File, ->
ACTION,MU,filemenu,QEND
...
QMENU
QID,filemenu,ITEMFILE,myfilemenu,COLS,6,ROWS,1,QEND
```

In the example, the menu header provides the label `File`, the `QID` name of the `QMENU` item is filemenu. The menu is in a file `myfilemenu`.

The width (`COLS` or `WIDTH` attributes) of the menu are set to be the same as the width of the `QMESSAGE` item. Note that a `ROWS` (or `HEIGHT`) attribute must be included, but it is not currently used.

The position at which a menu appears is determined by the position of the item that invokes the menu. The menu is displayed immediately following the `QMESSAGE` item. If the menu requires more rows than the space available within the window, the menu is truncated and the extra options cannot be accessed.

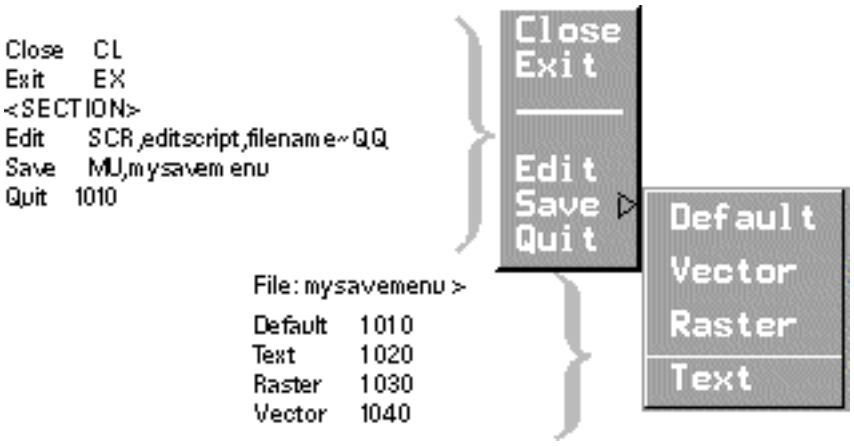
Walking Menus

Walking menus are menus in which the items, if selected, expand to provide additional options. This is achieved by creating further menus.

For example, in the preceding File menu, the user may need an option to save the file in one of a number of formats.

This can be implemented by creating a further Menu Item Definition File for the format option, and then specifying an `ACTION,MU` action which references this file as the command for the Save option in the first menu file. An arrow indicating that sub-options are available is automatically added by AccessWare.

Figure 3-9 Example of a Menu Item Definition File supporting Walking Menus



The new submenu is positioned to the right of the parent menu. The resource file to reference these menus would contain:

Example:

```
QMESSAGE
XPOS,20,YPOS,30,WIDTH,150,HEIGHT,25,LABEL,File,->
ACTION,MU,filemenu,QEND
...
QMENU
QID,filemenu,ITEMFILE,myfilemenu,COLS,6,ROWS,1,QEND
QMENU
QID,savemenu,ITEMFILE,mysavemenu,COLS,7,ROWS,1,QEND
```

Menus can be nested to any depth. If the menus expand such that they reach the right hand edge of the window, the menus stack one above the other.

Popup Menus

Popup menus can be provided for any `QLIST` item and invoked when the right mouse button is clicked. Popup menus can also supply a list of options for, say, a text input item. In this case they are invoked when the item is selected with the left mouse button.

A `QMENU` item and a Menu Item Definition File are created in the same way. They are associated with the item concerned via an `ACTION,MU` attribute specifying the `QID` of the `QMENU`.

Use with `QLISTS`: Popup menus can be used to provide a list of options to perform actions related to the list and items selected in it. These may include commands to update the list contents and/or commands to perform an operation on selected items. For example, if a list of files is displayed, actions may include Save, Print, etc.

Use with Text Fields: Popup menus are used to enter data into a text field. Each item in the list of options should invoke an appropriate action to update the text field contents.

Please note: Menus provide simple popups. Complex popup windows can be defined and referenced by a `QPOPUP` item.

Creating and Using Icons

Icons are used in interfaces to provide a graphical representation of an item or an option. They can make a user interface more interesting and reduce eye strain for the user.

The main advantage of icons is that the user can often recognize an item more easily than its equivalent word or abbreviation. By having no language constraints, icons also may be international, although this is not the main reason for using them.

Use of Icons in AccessWare

Icons can be used in AccessWare for `QWINDOW` and `QBUTTON` items.

QWINDOW Items

The icon associated with a `QWINDOW` is displayed when the window is closed.

For AccessWare running on most platforms, `QWINDOW` icons must be defined using the native window manager icon editor. Refer to the appropriate platform system manual for details.

If an AccessWare application is running under X windows (including OpenLook and DECWindows), AccessWare icons can be used by the `QWINDOW` item. The environment variable `AW_XICON_TYPE` must be set to '`aw`' if the `QWINDOW` icons are in AccessWare format, or to '`xbm`' if they are in the native X format.

If an AccessWare application is running on a PC (Win95/WinNT), AccessWare icons can be displayed in the `.bmp` format.

The `ICONLABEL` attribute, in the attribute list for the `QWINDOW` item, defines the label displayed below the closed window icon.

QBUTTON Items

In user interfaces, icons are frequently used on buttons to provide faster recognition of the command option offered. Also, an array of icon buttons across the screen can offer more options in a given space than buttons with text.

A significant advantage of an array of buttons, rather than a menu, is that a button immediately invokes the required option in a single operation, whereas a menu requires multiple clicks, first to display the menu and then to select an item from it.

Two icons can be defined for a QBUTTON item, to represent its selected and its unselected states. The ICON and DICON attributes respectively identify these icon files in the attribute list for a QBUTTON item. Refer Chapter 4, "AccessWare Item Definition Reference" for more information.

Creating an AccessWare Icon

Icons for use with QBUTTON, QLIST, and (where applicable) QWINDOW items are defined in an AccessWare icon text file. The format of this file is independent of the platform, host window system, and operating system under which AccessWare runs. Hence, the icons are portable.

Icons are rectangular, with the width and height defined in pixels using the WIDTH and HEIGHT attributes on the first non-comment line of the file. For example, to define an icon 16 pixels wide by 10 pixels high include the following line:

```
WIDTH,16,HEIGHT,10
```

The color of each pixel is defined by a hexadecimal number corresponding to the required color in the palette file. Each color defined in the color palette file is allocated a number between 0 and 15 and the hexadecimal equivalents, 0 - F, are used in the icon text file. For further information about the palette file see "Using Colors in AccessWare" on page 3-43. The following is an example of an icon text file.

```
#A sample icon - a multicolored X
      WIDTH,16,HEIGHT,8
      1100000000000099
      002200000000AA00
      0000330000BB0000
      00000044CC000000
      000000DD55000000
      0000EE0000660000
      00FF000000007700
      1100000000000088
```

The displayed position of each pixel on the screen corresponds to the position its hexadecimal number occupies in the icon text file (namely, line number and character position). If a character position in the icon file is left blank, the color of the corresponding pixel defaults to the background color.

Comment lines beginning with # can be included at the start of the icon file, but must precede the `HEIGHT` and `WIDTH` definition.

In the example, if the default background color is used, the ‘0’s are unnecessary. The file can then be written as:

```
Example:#An example icon - a multicolored X
      WIDTH,16,HEIGHT,8
      11             99
        22         AA
          33     BB
            44CC
            DD55
          EE     66
        FF         77
      11             88
```

The icon, shown in the following figure is a matrix 100 pixels wide by 50 pixels, with the background color 0 defined as white and the characters created with bands of the characters ranging from hexadecimal 1 to F and back to 1.

Figure 3-10 Example of an Icon

Using Colors in AccessWare

AccessWare enables you to control the color of windows and items in the user interface. Color can be used in an AccessWare user interface in many ways, for example:

- to distinguish separate areas of a window by displaying them in different colors or with a different colored border.
- to color items such as buttons in order to distinguish between different groups, or to make them more prominent if they provide special functions that should be highlighted to a user.
- to color text in order to highlight items or attach special significance, for example, different colored text can denote which text fields can and cannot be edited.

Color can also be used in icons. Color makes a user interface more interesting to use and reduces eye strain.

Defining the Color Palette to Use

The use of color in an AccessWare user interface is controlled by a color palette file. This is a text file containing a table of color numbers and definitions of the color corresponding to each number. To define the color of an item, the item definition in the resource file must refer to the appropriate color number.

The color palette for a user interface is defined by the `PALETTE` attribute. If specified, it must be included as an attribute to the main window. The pathname of the palette file can be a full or relative pathname:

Example:

```
PALETTE, palettes/my_palette.pal
```

A default palette file, named `access.pal` located in the AccessWare palettes directory, is supplied with the software release.

Creating a Color Palette File

Sixteen colors can be defined in a single color palette file for use with items in the interface and/or icons. Each color is defined in terms of the amount of red, green, and blue it contains from a level of 0 (none) to 255 (100%).

Thus, white is defined as 255 255 255 and black as 0 0 0.

Each entry in the palette file consists of:

- a color number: The number used in the resource files
- the RGB definition: The amount of red, green, and blue
- a label: The name used for the icon files

The color numbers range from 0 to 15 (or 0 to F in hexadecimal).

The format of the file follows:

```
Format: Number<Tab>Red<Tab>Green<Tab>Blue<Tab>Label
```

Example:

```
0     0     0     0     Background
1     255   0     0     Red
2     0     255   0     Green
3     0     0     255   Blue
4     255         255   Magenta
5     120   130   50    Mud
```

Please note: Several entries in the color palette file have special uses:

- Index 0 is the default background color
- Index 11 should be maintained as white
- Index 12 should be maintained as black
- Index 14 is any default foreground color
- Index 15 is the cursor color

The default colors used for various purposes are:

- Index 11 is the white color used for creating depth
- Index 12 is the black color used for creating depth
- Index 11 is also the default TEXTCOLOR
- Index 13 is the default BACKCOLOR
- Index 14 is the default ITEMCOLOR
- Index 3 is the default SELECTCOLOR

The following environment variables can be used to reset the preceding defaults:

- Itemcolor `AW_ITEM_COLOR`
- Textcolor `AW_LABEL_COLOR`
- Selectcolor `AW_SELECT_COLOR`
- Backcolor `AW_BACK_COLOR`

Please note: Set these environment variables to a value from your palette file. The prompt and message colors can be set separately by environment variables.

Providing Context Sensitive Help

AccessWare enables Help in the form of prompts, messages, and Help files to be provided by a user interface.

There are two ways in which you can provide context sensitive Help to a user:

- A text field can be defined as a prompt field, to display prompts:

  As the user moves the mouse cursor into fields or over buttons prompt messages can be displayed to the user in a specially defined Prompt field.

- A Help window can be defined, to display help files:

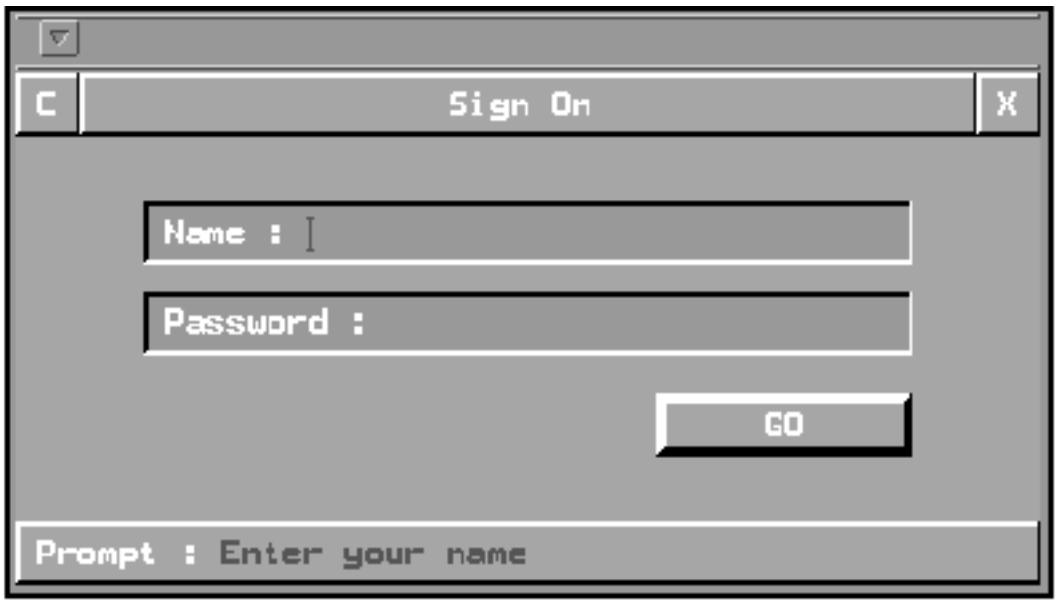
  The click of a Help button displays the cursor as a query symbol. Selecting an item in a window can display a scrollable piece of text information in a specially defined window.

A message area can also be defined in a window so that warnings, errors, or any other information can be displayed to the user.

Defining a Prompt Action

AccessWare provides the `PR` action to identify a field for the display of messages. This action should be applied to a suitably defined `QPANEL` item to display the Prompt text.

Figure 3-11 `QTEXT` **Item with PROMPT Attribute &** `QPANEL` **Item with ACTION,PR Attribute**



Only one prompt field should be defined for a particular window. The prompt field is placed towards the bottom of the window; it is often included with other footer items in a window. The preceding figure is created with an item definition as follows:

```
QPANEL
XPOS,0,YPOS,200,WIDTH,300,HEIGHT,25,LABEL,Prompt:,ACTION,PR,QEND
```

The prompt text for any item (for example, a QTEXT field) is specified using the PROMPT attribute in its definition.

Example:

```
QTEXT
QID,name,XPOS,10,YPOS,10,COLS,30,ROWS,1,INVERT,1, ->
LABEL,Name: ,PROMPT,Enter your name,QEND
```

When the user moves the mouse cursor into the item field, the prompt text defined for it is displayed in the QPANEL item for which the ACTION,PR attribute defined in the same window.

A user prompt is one of the simplest, most effective ways of providing the user with an indication of what is expected.

Creating a Message Field

AccessWare provides an ME action to identify a field for the display of messages from the application to the user. This action should be applied to a suitably defined QPANEL item.

Example:

```
QPANEL
XPOS,0,YPOS,200,WIDTH,300,HEIGHT,25,LABEL,Message:,ACTION,ME,QEND
```

The message delivered to the ACTION,ME panel is read from the file defined by the Environment variable AW_MESSAGE_FILE. It is read after the execution of every ACTION. This file can be created by a script that is executing an action.

A message can also be sent to the message file from an application program, using the procedure call aw_message.

Please note:

- The Prompt and Message fields are often put together at the foot of a window, defined in the footer section of the resource file.
- There are two environment variables: `AW_PROMPT_COLOR` and `AW_MESSAGE_COLOR`, which can be set to give the prompt and message text distinctive colors.

Setting Up a Help Window

To give a user more comprehensive information about an item or a function within your interface, you should provide a Help window. A Help window enables text files to be displayed in a scrollable list.

The HELP Window

The Help window is defined by separate resource file. The window must include one `QLIST` item, which displays the contents of the item's Help file in the Help window. The Help window for an application is defined by the `HELPWIN` attribute, which is included in the definition of the `QWINDOW` item of the main window resource file.

Example:

```
. . . HELPWIN,helpwin_qid . . .
```

Its argument specifies the `QID` of a `QPOPUP` item that represents the Help window. The `QPOPUP` item in turn identifies the resource file where the Help window is defined.

Within the resource file the definition for the `QLIST` item includes a `ITEMFILE` attribute that identifies a generic Help text file into which the contents of an item's Help file is copied for display.

The Help window can also include buttons, scrollbars, text fields, or any other items that are considered appropriate to its function for the particular application that you are developing. It is likely to include at least a vertical scrollbar for the list and buttons to close or quit the window.

Toggling the Help Action

The provision of Help is toggled on and off by selecting items that have an `ACTION` attribute of Help. The attribute is given to any item that toggles the Help facility. It is a toggle action, that is, selecting a Help item when Help is OFF switches it on and selecting a Help item when Help is ON switches it off. The Help item could be a menu option, or a button.

When the HELP is toggled ON, the following happens:

- The cursor changes to a query symbol to indicate Help is ON.

  If an item is now selected, the following occur:

- the contents of the selected item's `Help text file` is copied into a generic `Help text file` of the `QLIST` item in the Help window.
- the Help window is displayed, if it is not already open and visible.
- the contents of the `Help text file` are displayed in the `QLIST` of the Help window for the user to scroll through as required.

  The selected item's Help text file is identified by its `HELPFILE` attribute. The generic Help text file of the `QLIST` is identified by its `ITEMFILE` attribute.

A `HELP` action would be provided by one of the items in a dropdown Help menu or by Help button and made available for each window in the application. It can also be assigned to a Quit or Cancel button in the Help window. Thus Help can be switched on and off in various places.

Chapter 4

AccessWare Item Definition Reference

This chapter provides a reference guide to the items, attributes and actions used in the definition of an AccessWare user interface.

- Overview
- Window Item Reference
- Item Attribute Reference
- ACTION Attribute Reference

Overview

AccessWare allows the user to define and customize the Graphical User Interface (GUI) using specific items, attributes and actions. The significance of these definitions is explained as follows.

- Items

  In an AccessWare interface, items are defined to represent the text fields, buttons, lists, and menus, to be displayed and used in the interface.

  An item in the resource file is defined by a keyword that defines the type of item, followed by a list of attributes that describe its physical and logical characteristics, and the actions that it invokes.

- Item Attributes

  Items in a user interface are characterized by their attributes, which define its size, position, and visual effect. A considerable number of attributes may be given to an item in a window and the window can have a number of attributes.

  The attributes for an item are listed on the line following the item keyword. Attributes can take various types of argument, including integer and real numbers, text strings, file names, and recognized keywords.

- Actions

  Actions can be associated with most items in a window. An action forms the link between the user interface and the application. An action is introduced by the `ACTION` attribute.

  Actions are associated with buttons and initiated when a button is selected. However, actions may be linked with any item. Actions may also perform operations associated with the interface, such as updating a displayed list.

Window Item Reference

In AccessWare, window items comprise a range of different item types which provide the following functions:

Table 4-1 Brief Description of AccessWare Items

Item	Description
QWINDOW	Introduces a window definition and the attributes to the window item. It must appear at the beginning of each window resource file.
QPANEL	Defines a rectangular area within a window. It is used for visual effects and provides some specialized functions.
QBUTTON	Defines a button that the user can select for performing an immediate action.
QCHOICE	Displays a list of choice buttons for the user to select one or more options.
QMESSAGE	Provides a single line field in the window for displaying a string of text to the user.
QTEXT	Provides a text field complete with text editing functions for the user to enter data.
QLIST	Enables a list file to be displayed either as a scrollable list of text information or as a tree. Vertical and horizontal scrolling are defined independently.
QMENU	Defines a menu format, and references the name of the file providing a list of options for the menu contents.
QMENUITEM	Defines a single menu item.
QCONTROLSTATE	Controls the state of QMENUITEM. The control state can be visibly dependent or actively dependent.
QEXPRESSION	Defines the boolean of two control states.
QPOPUP	Provides a link to another window, by referencing the resource file of the new window.
QGROUP	References a list of items that form a group. Any action applied to the group affects all items in the group.
QTABCONTROL	Provides tab controls for a given window.
QMENUBAR	Generates top level menus for an application
QSEPARATOR	Generates a separation within a panel/window
QSLIDER	Generates a slide bar to adjust a numeric value

These items are described in detail in the following pages.

QWINDOW Item

The resource file for the user interface must contain a window definition and the attributes to the window item. It must appear at the beginning of each window resource file. A window is defined by its size, position, name. A window text font can be specified, as can the icon file to use if the window is closed.

The main window (the first window displayed) can also have additional attributes to define the Help window and color palette, switch the banner page on/off, and specify a global variable file.

Size and Position

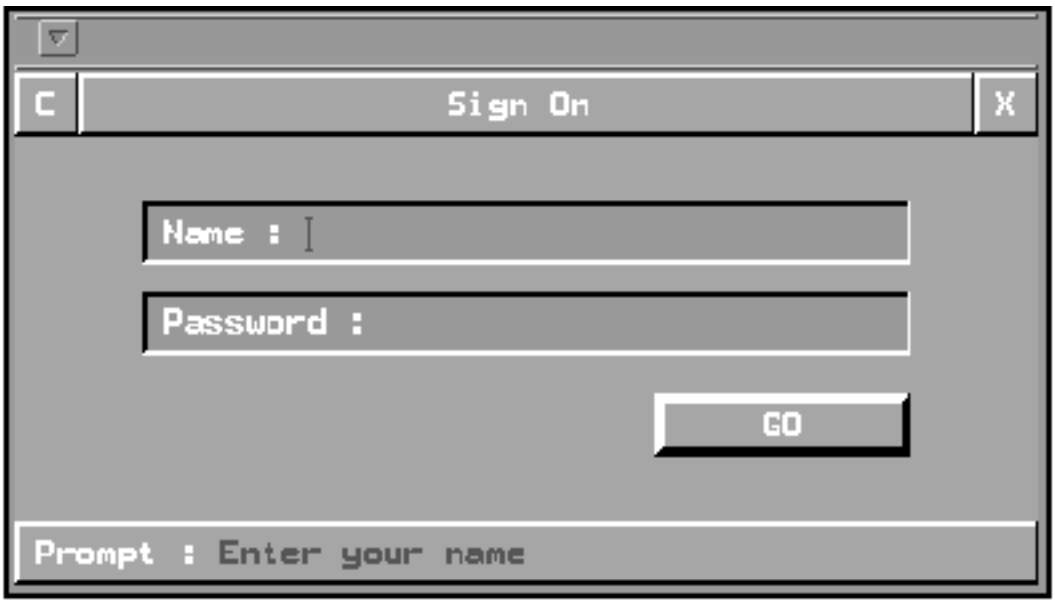
The window size and position attributes are obligatory, all other attributes can be set from defaults.

The keywords `XLOC` and `YLOC` define the position of the window. They displace its origin (its top left corner) by the number of pixels specified from the top left hand corner of the screen.

A window position of `XLOC,100,YLOC,200` places its origin 100 pixels from the left side of the screen and 200 pixels down from the top of the screen.

The size of the window is defined also in pixels using the keywords `WIDTH` and `HEIGHT`. The window shown in the following figure has a horizontal size of 400 pixels and vertical size of 200 pixels.

Figure 4-1 Example of a Window



The definition of this window in its resource file is as follows:

```
QWINDOW
QID,sign_on,XLOC,100,YLOC,200,WIDTH,400,HEIGHT,200,QEND
```

Please note: The width, height, and origin of a window are related to the usable area of the window, that is, the area inside the frame placed by the windowing system.

Definition of Items in a Window

Buttons, text fields, lists, and other items that appear within the window are defined in its resource file by their respective keywords. These keywords are described in the following pages.

QPANEL Item

The QPANEL item defines a rectangular area within the window. This is often used to divide the window into different logical areas, or provide a title for an information panel. The panel is frequently given a border to show the boundary of the area. For example, defining the ETCH item attribute for a panel makes it stand out.

The QPANEL item also has specialized uses, such as providing a header bar above a list, and scrollbars for scrollable lists. The panel is configured for these functions by ACTION attributes included in its definition. For example,

- ACTION,HEAD, defines a scrolling header bar.
- ACTION,VS, defines a vertical scrollbar.

Refer to "Scrollbars" on page 3-28 for more information.

A QPANEL item is also used to create prompt and message areas in the window, for example,

- ACTION,PR, defines a panel as a prompt field.
- ACTION,ME, defines a panel as an application message field.

No input is associated with a panel. However, it can be given a text label, which is displayed in its top left corner.

Example:

```
QPANEL
XPOS,10,YPOS,20,WIDTH,100,HEIGHT,40,LABEL,This is a Panel,QEND
```

Items can be nested, that is, placed one on top of another. For example, a text input field or button can be superimposed on a panel. The order that items appear in the resource file determines the sequence in which they are displayed. Hence, to place a button on a panel, the QBUTTON item definition must follow the QPANEL item definition. Otherwise, the button is hidden by the panel.

QBUTTON Item

This item defines a single button, used to initiate an action in the application. When the user clicks the button, the button appears pressed and then released again.

The button can be annotated with a central text label or with an icon. If an icon is used in conjunction with a text LABEL keyword, the text is displaced to appear below the icon.

Buttons are the most common method of implementing an action in the application.

Example:

```
QBUTTON
XPOS,100,YPOS,200,WIDTH,100,HEIGHT,50,LABEL,Print,ACTION,1001,->
QEND
```

QCHOICE Item

The QCHOICE item displays a list of options in button format. The type of format used depends upon which attributes appear in its definition.

The default presentation is a horizontal row of rectangular buttons, each centrally labelled with one option from the list. The default selection mode is single, that is, selecting one button automatically deselects any other.

The options list is defined using the LABEL attribute, with each option separated by a tilda character (~) and terminated by QQ.

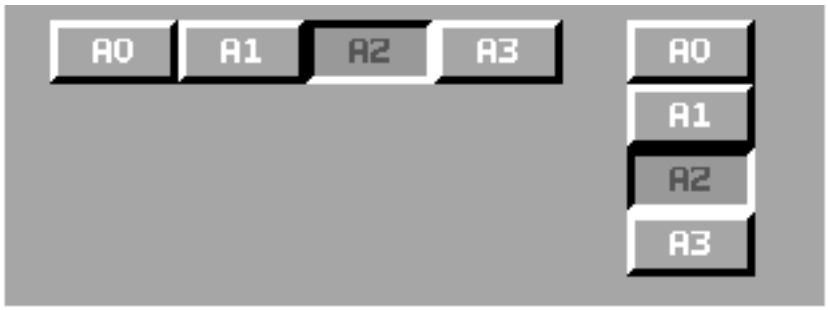
Example:

```
LABEL,A0~A1~A2~A3~QQ,
```

The VALUE attribute is used to specify a default selection as seen in the following figure. VALUE is followed by an integer representing the position of the required default option in the list. For example, to set the default in the preceding example to A2, the attribute definition is as follows:

```
VALUE,3,
```

Figure 4-2 QCHOICE **Items with** LAYOUT,H **&** LAYOUT,V **Attributes**



The LAYOUT attribute determines whether the buttons are arranged horizontally (LAYOUT,H) or vertically (LAYOUT,V), but in both cases they are juxtaposed as shown in the preceding figure.

The MODE,MULTI attribute specifies multiple-selection mode, that is, more than one button can be selected at a time.

The STYLE,TOGGLE attribute changes the size and shape of the buttons to toggle (or radio) buttons and displays their labels to the right of each button as seen in following figure. The shape of the buttons depends upon the mode as follows:

- Small diamond-shaped buttons for single-selection mode
- Small square buttons for multiple-selection mode

Figure 4-3 QCHOICE **Items Using** STYLE,TOGGLE **and** MODE,MULTI

Example:

```
QCHOICE
XPOS,100,YPOS,200,WIDTH,40,HEIGHT,25,LABEL,A0~A1~A2~A3~QQ, ->
VALUE,2,STYLE,TOGGLE,LAYOUT,H,QEND
```

QMESSAGE Item

This is a simple field that is used to display a message. It is a simpler form of the QTEXT field type, but permits only a single row and does not accept any input from the user or the application. It displays only the text defined by the LABEL attribute.

Please note: This field is not accessible to the user.

QMESSAGE items can be used to provide dropdown menus from a menu bar. Each option keyword in the menu bar is a QMESSAGE item with an ACTION,MU attribute.

Example:

```
QMESSAGE
XPOS,10,YPOS,25,WIDTH,75,HEIGHT,25,LABEL,File,ACTION,MU,QEND
```

QTEXT Item

The QTEXT item provides a field that accepts user typed input, which is displayed in the field and can be edited. The field can be defined to be a number of rows, provided that the height of the field is defined using the ROWS attribute and

MODE,MULTI is specified. The input text automatically wraps from one row to the next.

Please note: If the height of a QTEXT item is defined using the HEIGHT attribute instead, the field is displayed with the specified height. However, regardless of the height of the field, any input text is displayed as only a single row of text vertically centered in the field and truncated at the extremity of the field width specified.

A short text prompt can be defined at the start of the field to indicate what input is expected. The required prompt text is defined using the LABEL attribute. The text cursor is then positioned after the prompt.

Also, a default text string can be inserted in the field by using the VALUE attribute. The user can accept the default or edit it.

Example:

```
QTEXT
XPOS,50,YPOS,50,WIDTH,300,HEIGHT,25,INVERT,1,LABEL,Name: ,QEND
```

Any printable characters are considered valid input. Any data validation must be performed by the application.

Please note: For information regarding cut, copy, paste, and drag-and-drop operations, see “QTEXT Item” on page 3-19.

Please note: For information on creating a link between a textbox and a list box, see “QLIST Item” on page 3-16.

QLIST Item

The QLIST item displays a list of data from which the user can select one or more rows. The QLIST definition must include the ITEMFILE attribute to identify the name of the file containing the contents of the list.

The data can be displayed either as a text-based listing, or, if a suitably formatted ITEMFILE has been defined as a tree.

By default, a QLIST item operates in single-selection mode, whereby only one row can be selected at a time. The attribute MODE,MULTI specifies multiple-selection mode, where more than one row can be selected at a time.

The list scrollbars are independently defined as QPANEL items with special AccessWare standard built-in actions to provide the required scrolling functions.

A non-action panel item can be positioned above a list to provide a list title. Alternatively, a header bar that scrolls with the list, keeping the header columns in line with the list columns can be defined.

A scrolling header bar is a separately defined QPANEL item that is configured to maintain vertical alignment with the list by an ACTION,HEAD attribute. It also obtains the header line from a file in the same directory as the list contents file, enabling it to be dynamically created in conjunction with the list contents file.

A list can also have an associated popup menu that enables commands to be activated for options in the listing.

Example:

```
QLIST
QID,dwg_list,XPOS,50,YPOS,50,WIDTH,300,HEIGHT,200, ->
ITEMFILE,/usr/lists/drawing_list,QEND
```

Instructions on defining lists to include scrollbars and scrolling header bars, and lists displayed as trees, are given in the sections “Working with Lists” on page 3-27 and “Displaying Trees in AccessWare” on page 3-32.

Please note: For information on creating a link between a textbox and a list box, see “QLIST Item” on page 3-16.

QMENU Item

The QMENU item displays a list of options in a menu format. It must include the ITEMFILE attribute to identify a text file (the Menu Item Definition File), which contains the list of menu options and their actions.

If an action is not specified for an option in the file, or an option is not selected, a default ACTION specified in the QMENU definition is initiated instead.

The COLS or WIDTH attribute defines the width of the menu. The width should be set to accommodate the longest text option. A height attribute must be specified for a QMENU item, although AccessWare automatically adjusts the menu height to fit the number of options.

The colors and text fonts for the displayed menu can be specified as separate attributes in the QMENU definition.

Example:

```
QMENU
QID,menu1,COLS,8,ROWS,6,ITEMFILE,/menu1,DEPTH,1,QEND
```

The position of QMENU items is preset by AccessWare and cannot be specified manually.

A menu option can be expanded into a series of sub-menus (walking menus). Each sub-menu is defined as a separate QMENU item. A sub-menu can be invoked when an ACTION,MU is defined in its Menu Item Definition File.

For further information on defining menus, refer to "Working with Menus" on page 3-35.

QMENUITEM Item

The QMENUITEM item defines a single menu item of a menu such a FILE > NEW where FILE is the menu and NEW is the menu item. Each menu item is associated with an ACTION and it can have control state objects using ACTIVEDEP and VISIBLEDEP attributes.

Example 1:

```
QMENUITEM
QID,CA_MENUID_VAULT,ACTIVEDEP,CA_STATE_SIGNON,LABEL,->
$CA_VAULT_LABEL,VISIBLEDEP,CA_EXPR_STD_OR_SAP_OR_CM_OR_CAMU
```

In the preceding example, the values of ACTIVEDEP and VISIBLEDEP are QIDs control states for a given item. The control state indicates visible or active dependency. ACTIVEDEP will show the item disabled or enabled depending on the attribute value. VISBLEDEP will show or hide the item depending on the value specified in the QCONTROLSTATE item.

Example 2:

```
QMENUITEM
QID,CM_MENUID_CLOSE_ASSM_CAMU,VISIBLEDEP,CA_STATE_CM_MODE,->
ACTIVEDEP,CM_STATE_SIGNON,LABEL,$CM_MENUID_CLOSE_ASSM_CAMU_LBL,->
ACTION,6501,UDATA,$CM_MENUID_CLOSE_ASSM_CAMU_LBL   6501   222,QEND
```

The preceding example shows how UDATA is used in a QMENUITEM definition. For more information on UDATA, see "UDATA" on page 4-47.

Example 3:

```
QMENUITEM
QID,CM_MENUID_FILTERS,VISIBLEDEP,CA_STATE_CM_MODE,ACTIVEDEP,CM_ST
ATE_SIGNON,LABEL,$CM_MENUID_FILTERS_LBL,ACTION,SY,$AW_PERL_EXEC
$CM_SCRIPTS/setfilt OPEN &,QEND
```

The preceding example executes a script in a similar manner using the previous QMENUITEM definition.

Please note: The menu is defined by the QMENU item, but the items within a menu are defined by the QMENUITEM item using this style of menus. The old format, ordinary menu definitions can also be used.

QCONTROLSTATE Item

The QCONTROLSTATE item is defined for each occurrence of ACTIVEDEP and VISIBLEDEP in QMENUITEM definitions. The QID of QCONTROLSTATE item is used as the value for the ACTIVEDEP and VISIBLEDEP attributes specified in the QMENUITEM item.

Example:

```
QCONTROLSTATE
QID,CA_STATE_SYSADMIN_SIGNON,VALUE,0,QEND
```

QEXPRESSION Item

The QEXPRESSION item is used to define the boolean of two control states. For each menu item, if this expression is identified then menu items can depend on this expression instead of the control state. Two or more control states can be used to form a boolean expression.

Operators supported for this expression object are && (and) and || (or).

Example: If the control states are:

```
QCONTROLSTATE
QID,CA_STATE_SYSADMIN_SIGNON,VALUE,0,QEND

QCONTROLSTATE
QID,CA_STATE_PROJADMIN_SIGNON,VALUE,0,QEND
```

Then, the expression formed with the || (or) of these control states is as follows:

```
QEXPRESSION
QID,SYSADMINORPROJADMIN,VALUE,0,RULES,->
CA_STATE_SYSADMIN_SIGNON || CA_STATE_PROJADMIN_SIGNON,QEND
```

The following is an example of a QEXPRESSION defined using the && operator.

```
QEXPRESSION
```

```
QID,SAP_STATE_VLT_SAP_SIGNON,VALUE,0,RULES,SAP_STATE_SIGNON &&
CA_STATE_SIGNON,QEND
```

The following is an example of a QEXPRESSION with more than two control states.

```
QEXPRESSION
QID,CA_EXPR_STD_OR_SAP_OR_CAMU,VALUE,1,RULES,CA_STATE_STD_MODE ||
CA_STATE_SAP_MODE || CA_STATE_CAMU_MODE,QEND
```

QPOPUP Item

The QPOPUP item identifies another window in the interface, which is used as a popup window for display with the current window, or to which control is transferred. The QPOPUP item provides the link by referencing the resource file of the new window.

The definition includes a window name, a visibility setting (that is, whether displayed or hidden on starting the application).

The LABEL keyword specifies the resource file pathname.

Example:

```
QPOPUP
QID,new_win,VISIBLE,0, ->
LABEL,/usr/resource_files/new_window.res,QEND
```

Please note: The QPOPUP items are the items that link all the windows to each other in an AccessWare application interface.

QGROUP Item

The QGROUP item specifies a group of items that are treated as one item. Any ACTION applied to the QGROUP affects all items in the group.

The ITEMFILE attribute identifies the file name containing the QIDs for all items included in the group. The format of the file follows:

```
qid1
qid2
...
qidn
<EOF>
```

Actions that can be applied to the group as a whole are as follows:

`ACTIONs SH, HI, GO, GI` (and the equivalent procedure calls). For more information on these actions refer to “ACTION Attribute Reference” on page 4-54.

Example:

```
QGROUP
QID,group1,ITEMFILE,/usr/grp/my_group,QEND
```

Please note: Any actions defined individually for items included in the group do not affect the other items in the group.

QTABCONTROL Item

The `QTABCONTROL` item defines tab controls for a given window. This item is used to divide the window into different logical areas and provide a title for each logical area. The panel can be given a border using the `ETCH` keyword to show the boundary of the area.

The `QTABCONTROL` item also provides a header bar above a list and scrollbars for scrollable lists. The tabs are configured for these functions by the action attributes included in its definition.

- `ACTION,HEAD,` defines a scrolling header bar.
- `ACTION,VS,` defines a vertical scrollbar.

Refer to “Scrollbars” on page 3-28 for more information.

A `QTABCONTROL` item is also used to create prompt or message areas in the window.

- `ACTION,PR`: Defines a panel as a prompt field.
- `ACTION,ME`: Defines a panel as an application message field.

No input is associated with a panel. However, it can be given a text label, which is displayed in its top left corner.

Example:

```
QTABCONTROL
XPOS,10,YPOS,20,WIDTH,200,HEIGHT,20,->
LABEL,General~Attributes~QQ,VALUE,1,QEND
```

QMENUBAR Item

The QMENUBAR item defines top level menus for an application. The menubar enables the user to create items for a pulldown menu.

Example:

```
QMENUBAR
XPOS,0,YPOS,0,WIDTH,100,HEIGHT,25,DEPTH,2,ITEMFILE,somefile,QEND
```

An ITEMFILE in the previous definition would contain the list of items for the menubar and its associated Menu Item Definition File. See "Menu Item Definition Files" on page 7-17.

QSEPARATOR Item

The QSEPARATOR item generates a separation within a panel or window. This item creates a logical area within your dialog box. For example, you could separate different application functions within a dialog box.

Example:

```
QSEPARATOR
XPOS,10,YAFTER,2,WIDTH,150,HEIGHT,148,DEPTH,1,QEND
```

QSLIDER Item

The QSLIDER item generates a slide bar to adjust a numeric value. Specify the maximum and minimum values using the MAXVALUE and MINVALUE attributes. The following ACTION is associated with the slider bar.

- ACTION, programmatic_action: Specifies the program to be run when the slider is moved.

Example:

```
QSLIDER
QID,xsliderr,ETCH,4,XPOS,0,YAFTER,0,WIDTH,130,HEIGHT,25, ->
LAYOUT,H,LABEL,0~100~QQ,VALUE,10,ACTION,1001,MINVALUE,0, ->
MAXVALUE,100,VISIBLE,1,QEND
```

QATTREND Keyword

The QATTREND keyword terminates each resource file and denotes the end of the window specification.

Item Attribute Reference

An item can have a large number of attributes, some of which are common to the majority of items, for example, those defining size and position. However, other attributes apply only to an individual item or a limited range of items. In addition, although the same attribute keyword can be used with a number of different items, the function it provides and the arguments it takes often vary from item to item.

Please note: In a resource file the following statements apply:

- All attributes must appear on the line immediately following the item type keyword. Blank lines and comment lines are only permitted between (not within) the definitions of individual items.
- All attributes for an item must be entered on a single line in the file.
- Commas must be used to separate the attribute keyword, the attribute arguments, and other attributes. Spaces are not permitted, except in text strings.
- The order in which the attributes are defined is not significant.
- The definition of each item is terminated by the keyword `QEND`.

There is a limit of 1000 characters in the argument list to an attribute.

Item Attributes Summary

The item attributes can be categorized in functional groups as follows:

Naming Items	
`QID`	Specify an item identifier
Attributes to Position Items	
`XLOC,YLOC`	Specify the location of a `QWINDOW`
`XPOS,YPOS`	Specify position of item in a window
`XOFFSET,YOFFSET`	Position item relative to another item
`XAFTER,YAFTER`	Position item relative to another item
Attributes to Define the Size of Items	
`WIDTH,HEIGHT`	Specify the size in pixels
`MINWIDTH`	Minimum width of a window in pixels during creation or resize.
`MINHEIGHT`	Minimum height of a window in pixels during creation or resize.
`MAXWIDTH`	Logical maximum width of item in pixels
`MAXHEIGHT`	Logical maximum height of item in pixels
`COLS,ROWS`	Specify size with respect to item text
`MAXCOLS`	Logical maximum width of item in terms of characters

MAXROWS	Logical maximum height of item in lines of text

Attributes to Specify Behavior of an Item When Resized

MOVEX	Percentage of resize to move item in the X direction
MOVEY	Percentage of resize to move item in the Y direction
STECTCHX	Percentage of resize to stretch item in the X direction
STRECTCHY	Percentage of resize to stretch item in the Y direction

Attributes to Define the Visual Style of an Item

ACTIVE	Set item initially active or inactive
VISIBLE	Make the item visible/invisible
ACTIVEDEP	Set menu item active or inactive as defined in QCONTROLSTATE or QEXPRESSION
VISIBLEDEP	Set menu item visible or hidden as defined in QCONTROLSTATE or QEXPRESSION
DEPTH,INVERT	Give a 3-dimensional appearance
ETCH	Give item an etched-look border
STYLE	Set the presentation style for an item
LAYOUT	Orientate QCHOICE option buttons
ICON,DICON	Associate icons with an item
ICONLABEL	Specify a label for a closed window

Attributes to Control the Color of Items

ITEMCOLOR	Specify the color of an item
TEXTCOLOR	Specify the text color of an item
BACKCOLOR	Specify color of a selected item
SELECTCOLOR	Specify text color of selected item
GREYOUTCOL	Specify a different grey-out color

Attributes to Assign Labels to the Item

LABEL	Specify a text label

Attributes to Define the Field Contents

LISTMENU	Supply a popup menu for a QLIST
ITEMFILE	Reference list contents of a text file
LISTFILE	Same as ITEMFILE (Use ITEMFILE instead.)
MINVALUE	Minimum value of an object where applicable
MAXVALUE	Maximum value of an object where applicable

Attribute to Supply Initial Defaults to an Item

VALUE	Set initial default values

Attributes to Assign Special Characteristics

STATE	Set special characteristics for item
MODE	Set select option/text display mode

DEFAULT	Mark an item as the default item to action when the ENTER/RETURN key is pressed
FOCUS	The item will take the keyboard focus when the window is shown
OVERVIEW	For a tree, overview another tree item
INTERVAL	For objects that can have a range of values, specify the interval by which they increment. See QSLIDER.

Miscellaneous Attributes for Items

FONT	Specify the text font for an item
RULES	Define the boolean of two or more control states
UDATA	Define user data

Attributes to Invoke the Help Facilities

PROMPT	Provide user prompts for the item
HELPFILE	Supply the name of a helpfile
HELPWIN	Specify the name of the Help window
TIP	Specify quick information about the button action in a tooltip

Attributes that Affect the Whole Application

RESIZE	Specific direction in which to resize a window
SPLASH	The name of a image to use as a splash screen
NOBANNER	Suppress a banner page at startup
NOSTRIPE	Do not include the window stripes
PALETTE	Specify the color palette
GLOBVARS	Supply file of environment variables.

Naming Items

A name, in the form of a QID identifier, can be allocated to any item defined in a window resource file. The application code and the other items in any of the windows use this item identifier when specifying any actions affecting it. If an item is not referenced by any other item, no QID is necessary.

The QID is thus the reference used by AccessWare ACTIONs and procedures to set values for the item, and hide or deactivate the item.

Please note: A QPOPUP item references a new window, and the QID of the QPOPUP is the identifier of that new window. However, the main window cannot be referenced by a QPOPUP. Instead, the special QID of mainwin is used to reference it.

QID

The QID attribute assigns the name by which other items and the application code identify the item.

The name must be an alphanumeric string up to a maximum of 63 characters. For example, a text input field to hold a user's password may be named user_passwd.

Example:
```
QID,user_passwd,
```

An item name must be unique across all items in all windows used by the application. Consequently, use standard naming conventions. To distinguish two similar items in different windows prefix the item name with a window ID. For example, QID,window1_Start, and QID,window2_Start, distinguishes two Start buttons.

There is no default. The item remain unnamed. The QID is applicable to all except the QWINDOW.

Attributes to Position Items

The position of a window is defined in pixels relative to the top left corner of the screen. The position of an item within a window is defined in pixels relative to the origin of the window or relative to another item.

Item positions must be specified for both X and Y axes. Consequently, a pair of the following attributes must be specified, although a combination of the different types can be used as the pair of X and Y.

Please note: Size and position attributes are required for all items except QPOPUP and QGROUP.

XLOC, YLOC

Use these attributes to position QWINDOW's only. This is the only method of positioning a window. They define in pixels the position of the window origin from the top left corner of the screen. Positive X direction is towards the right of the screen and positive Y direction is towards the bottom of the screen.

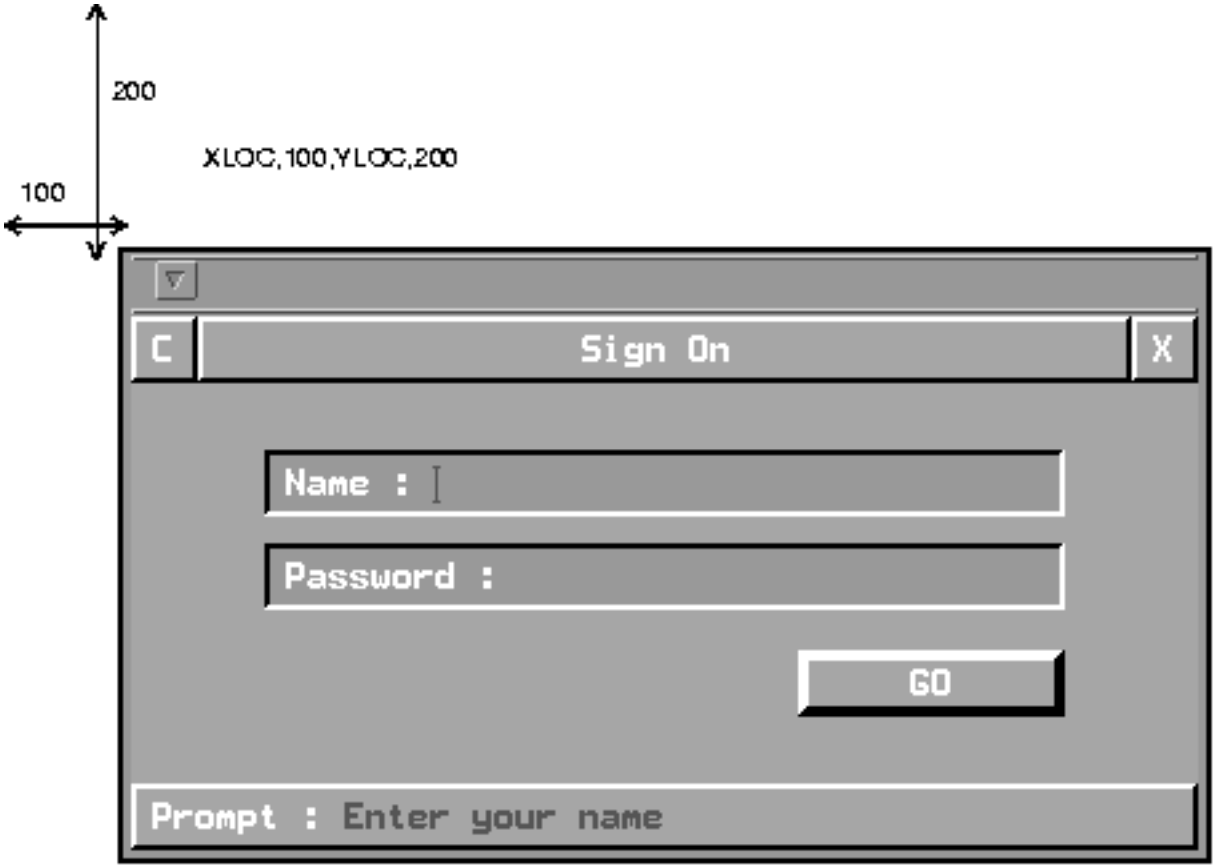
Example: To position a window at 100 pixels from the left of the screen and 200 pixels down from the top, define the attributes as follows:

```
XLOC,100,YLOC,200,
```

The following figure illustrates this example.

Please note: These coordinate positions apply to the usable area of the window. The window surround, which the applicable window system includes by default (and may differ from system to system), is outside the window position specified.

Figure 4-4 Window at `XLOC,100,YLOC,200`



There is no default `XLOC` or `YLOC`. Both these attributes are applicable the `QWINDOW` item alone.

XPOS, YPOS

These attributes define the position of an item within a window, or the location of the window icon when it is closed.

Item Position: The position of any item in a window is defined in terms of pixels from the top left corner of the window. Positive X direction is towards the right of the window, positive Y direction is towards the bottom of the window.

To position an item at 100 pixels from the left of the window and 200 pixels down from the top of the window, define the attributes as follows:

```
. . .XPOS,100,YPOS,200,. . .
```

There is no default XPOS or YPOS. Both these attributes are applicable to all items except QPOPUP, QGROUP and QMENU. For the QWINDOW item these attributes apply to a closed window icon.

Window Icon Location: Use these attributes for a QWINDOW item to define the position at which the icon is displayed if the window is closed. The X-Y values given specify the position of the icon in pixels from the top left hand corner of the screen.

Example:

```
XPOS,0,YPOS,0,
```

These attributes would place the icon for a closed window at the top left corner of the screen.

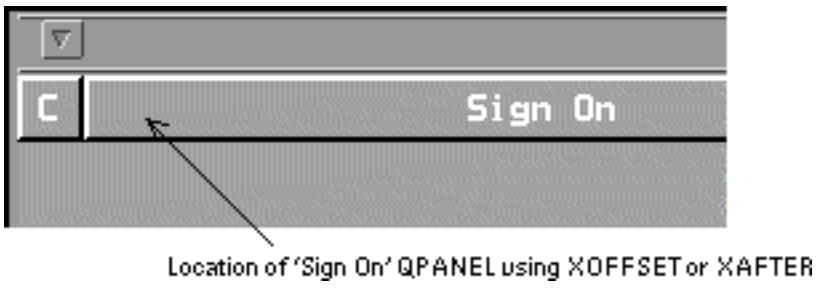
XOFFSET, YOFFSET

Instead of defining absolute pixel co-ordinates, you can position items relative to the previous item defined in the file.

These attributes define the position of an item as a specified number of pixels from the top left corner of the previous item in the file. A negative number of pixels defines a position to the left or above the previous item. The following is an example of a QPANEL item set relative to a button.

```
QBUTTON
XPOS,0,YPOS,0,WIDTH,25,HEIGHT,25,LABEL,C,ACTION,CL,QEND
QPANEL
XOFFSET,25,YOFFSET 0,WIDTH,450,HEIGHT,25,LABEL,Sign On,QEND
```

Figure 4-5 Example of a Sign On Panel Offset from Button



This example positions a panel called Signon immediately to the right of a button labelled C (that is, 25 pixels from its top left corner), as shown in the preceding figure.

There is no default XOFFSET or YOFFSET. Both these attributes are applicable to all items except QWINDOW, QPOPUP and QGROUP.

XAFTER, YAFTER

These attributes are similar to the offset attributes, but define the position of the item as a specified number of pixels from the bottom right corner of the previous item in the file.

Example:

```
QBUTTON
XPOS,0,YPOS,0,WIDTH,25,HEIGHT,25,LABEL,C,ACTION,CL,QEND
QPANEL
XAFTER,0,YOFFSET 0,WIDTH,450,HEIGHT,25,LABEL,Sign On,QEND
```

This example provides an even more useful way of positioning the Signon panel shown in Figure 4-5 to immediately follow the button.

The XAFTER,YOFFSET attribute combination used in this example. XAFTER,0 aligns the left edge of the panel with the right edge of the button and YOFFSET,0 aligns the top of the panel with the top of the button.

These keywords eliminate much of the calculation otherwise required to position items exactly. They enable relative positioning of items rather than absolute positioning. This is useful when modifying window layouts because the AFTER attributes require no changes.

There is no default XAFTER or YAFTER. Both these attributes are applicable to all items except QWINDOW, QPOPUP and QGROUP.

Please note:

- In defining the location of an item, ensure that it remains within the boundary of the usable window.
- Ensure also that a window remains within the area of the screen, taking note that the surround is not included in the position or size definition.
- A position attribute is required for all items other than QPOPUP and QGROUP.

Attributes to Define the Size of Items

Item size defines the area of the window used by the item or the size of the window. All items except QPOPUPs and QGROUPs must have size attributes. For QWINDOW items, the size specifies the usable area of the window and does not include the surround supplied by the window system.

Size is defined as numbers of pixels. However, for text items, size can also be defined as the numbers of rows for height and columns of characters for width. Thus, the ultimate size of the text item varies according to the font used.

WIDTH, HEIGHT

The width of an item is specified as the number of pixels that an item occupies in the horizontal (X) direction. This is the physical width of the item.

The height of an item is specified as the number of pixels that an item occupies in the vertical (Y) direction. This is the physical width of the item.

There is no default WIDTH or HEIGHT. Both these attributes are applicable to all items except QPOPUP and QGROUP.

Example:

```
WIDTH,100,HEIGHT,25
```

MINWIDTH, MINHEIGHT

The minimum width of a window is specified as the number of pixels that an item occupies in the horizontal (X) direction. This width is in effect during creation or resize.

The minimum height of a window is specified as the number of pixels that an item occupies in the vertical (Y) direction. This width is in effect during creation or resize.

There is no default MINWIDTH or MINHEIGHT. Both these attributes are applicable to the QWINDOW item alone.

Example:

```
MINWIDTH,100,MINHEIGHT,25
```

MAXWIDTH, MAXHEIGHT

The logical maximum width of a window is specified as the number of pixels that an item occupies in the horizontal (X) direction. This width is in effect during creation or resize.

The logical maximum height of a window is specified as the number of pixels that an item occupies in the vertical (Y) direction. This height is in effect during creation or resize.

Example:

```
MAXWIDTH,100,MAXNHEIGHT,25
```

COLS, ROWS

You can also specify the width and height of a text item as the number of columns (that is, columns of characters) and number of text rows it occupies. This is particularly useful for fixed length `QTEXT` input fields.

AccessWare calculates the width dimension for the field from the character width of the text font specified for the window. However, you should specify only fixed-width fonts. It also calculates the height dimension for the field from the text font specified for the window. This dimension is a function of the character height and line spacing of the font specified.

Example:

```
COLS,10,ROWS,4,
```

This example defines a text field as 10 columns (that is, characters) wide by 4 rows deep. `COLS` and `ROWS` settings override any `WIDTH` and `HEIGHT` modifiers for the item.

Please note: The `COLS` attribute defines the size limit for `QMENUs`. AccessWare will truncate any fields longer than the `COLS` or `WIDTH` dimension specified. However, although a height attribute (`HEIGHT` or `ROWs`) must be specified for `QMENU` items, AccessWare calculates the actual height of the menu to fit the number of options it contains.

There is no default `COLS` or `ROWS`. Both these attributes are applicable to all items except `QWINDOW`, `QPOPUP` and `QGROUP`. The `ROWS` attribute, however, not applicable to `QBUTTON`, `QCHOICE`, and `QMESSAGE` as well.

MAXROWS, MAXCOLS

You can also specify the maximum rows and columns of a text item as the number of characters (row of characters) and number of lines it occupies. This is particularly useful for fixed length `QTEXT` input fields. See also the previous section.

Example:
```
MAXROWS,10,MAXCOLS,4,
```

Attributes to Specify Behavior of an Item When Resized

MOVEX

The `MOVEX` attribute specifies the percentage the item is resized as it is moved in the X direction.

MOVEY

The `MOVEY` attribute specifies the percentage the item is resized as it is moved in the Y direction.

STRETCHX

The `STRETCHX` attributes keeps the distances between items equal as you resize a window in the horizontal direction.

STRETCHY

The `STRETCHY` attributes keeps the distances between items equal as you resize a window in the vertical direction.

Attributes to Define the Visual Style of an Item

There are a number of attributes that affect the appearance of an item.

Please note: The attributes that control color are dealt with separately. They are described in “Attributes to Control the Color of Items” on page 4-33.

ACTIVE

The `ACTIVE` attribute determines whether or not an item is initially in the active state when the application starts.

The ACTIVE attribute takes an argument of either 0 (inactive) or 1 (active).

For example, setting a Print button in a Print window to inactive until some information is entered is often desirable. By defining the Print QBUTTON item with an ACTIVE attribute set to 0, the button has no effect until the user performs some other operation.

Example:

```
ACTIVE,0,
```

The default value for this attribute is 1. It is applicable to all items except QWINDOW, QPOPUP and QGROUP.

VISIBLE

Any item (field, button, or window) can be made visible or invisible when the window is initially displayed. The application may change the visibility of an item to make it accessible or inaccessible according to the operational sequence at any instant.

The single argument should be set to 0 or 1, where an argument of 1 denotes a visible item (the default). To set an item invisible on starting up the application, set the attribute as follows:

```
VISIBLE,0,
```

This attribute is applicable to all items except QWINDOW, QMENU and QGROUP.

ACTIVEDEP

The ACTIVEDEP attribute represents the QID of the QCONTROLSTATE item based on whose value the menu item is active (not grayed-out) or inactive (grayed-out). A QCONTROLSTATE item is defined for each ACTIVEDEP occurrence in a QMENUITEM definition.

Example:

```
QMENUITEM
... ACTIVEDEP,CA_STATE_SIGNON ...
```

In the preceding example, the ACTIVEDEP attribute is defined for the CA_STATE_SIGNON menu item. Therefore, the QCONTROLSTATE item definition for the same menu item is as follows:

```
QCONTROLSTATE
QID,CA_STATE_SIGNON,VALUE,0,QEND
```

The VALUE attribute takes an argument of either 0 (inactive) or 1 (active).

VISIBLEDEP

The VISIBLEDEP attribute represents the QID of the QCONTROLSTATE item based on whose value the menu item is visible or hidden. The QCONTROLSTATE item is defined for each VISIBLEDEP occurrence in a QMENUITEM definition.

Example:

```
QMENUITEM
... VISIBLEDEP,CA_STATE_CM_MODE ...
```

In the preceding example, the VISIBLEDEP attribute is defined for the CA_STATE_CM_MODE menu item. Therefore, the QCONTROLSTATE item definition for the same menu item is as follows:

```
QCONTROLSTATE
QID,CA_STATE_CM_MODE,VALUE,1,QEND
```

The VALUE attribute takes an argument of either 0 (visible) or 1 (shown).

DEPTH,INVERT

Giving an impression of depth to a field (that is, a 3D effect to simulate a raised or a recessed item) can make the function or boundary encompassing certain items more obvious to the user. A button that can be pressed, or a text field available for input are typical examples.

The single argument of the DEPTH attribute is a positive integer specifying the depth (the height the item appears to protrude from the surface) in pixels.

The INVERT attribute is related to the DEPTH attribute. It produces a reverse 3D effect to DEPTH, thus simulating a recessed item. The INVERT attribute takes an argument of 0 or 1, where 1 inverts the DEPTH settings to produce the recessed effect.

DEPTH is more suited to show items such as unselected buttons, whereas DEPTH modified by INVERT is preferable for items such as input fields.

Set a button for a simulated depth of 5 pixels as follows:

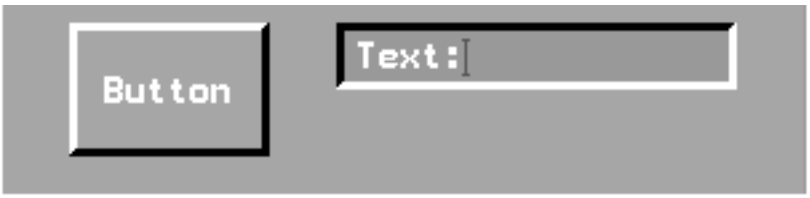
```
DEPTH,3,
```

To recess a field, set the INVERT attribute to 1, as follows:

```
DEPTH,3,INVERT,1,
```

The following figure illustrates the DEPTH and INVERT attributes.

Figure 4-6 Example Items with DEPTH and INVERT Attributes



The default for the DEPTH attribute is 2. The default for the INVERT attribute is 0. The INVERT attributes are applicable to QPANEL, QBUTTON, QTEXT and QLIST. The DEPTH attribute, however, is only applicable to QCHOICE, and QMENU as well.

ETCH

For consistency with the MOTIF style of interface, you can surround an item (for example, QBUTTON, or QPANEL) with an etched border.

The argument to the ETCH attribute defines the width of the etched border. The depth of etching is preset. The ETCH attribute can be used with INVERT to produce a raised etched border. The DEPTH attribute can also be used in conjunction with ETCH to give height to an item. The following figure is an example of items defined with ETCH, INVERT, and DEPTH attributes.

Example:

```
DEPTH,3,ETCH,5,
```

Figure 4-7 Example Items with DEPTH, INVERT, & ETCH Attributes

The default is no etching. It is applicable to QPANEL, QBUTTON and QLIST.

STYLE

The STYLE attribute is used with QCHOICE and QBUTTON items, to define or control how an item is displayed. The way in which the attribute is used depends upon the item type.

For a QBUTTON, the STYLE attribute takes an argument of UPARROW, DOWNARROW, RIGHTARROW, or LEFTARROW to define the orientation of the arrow button.

Example:

```
... STYLE,UPARROW ...
```

For a QBUTTON item, the UPARROW, DOWNARROW, RIGHTARROW, or LEFTARROW keywords are the only valid arguments to the STYLE attribute. The STYLE attribute is used with buttons that scroll lists.

For QCHOICE items, the STYLE attribute controls the way in which the list of option buttons are displayed. By default, a QCHOICE list is displayed as a set of rectangular buttons arranged horizontally, with button labels centered inside each button. The following figure illustrates the default presentation of a QCHOICE item.

Figure 4-8 Default Presentation of a QCHOICE Item

The TOGGLE argument to the STYLE attribute modifies this default by positioning the button labels to the right of each button and shows the buttons as small radio toggle buttons. The following figure illustrates a QCHOICE item with the STYLE,TOGGLE attributes defined.

Figure 4-9 QCHOICE Item with STYLE,TOGGLE Attribute

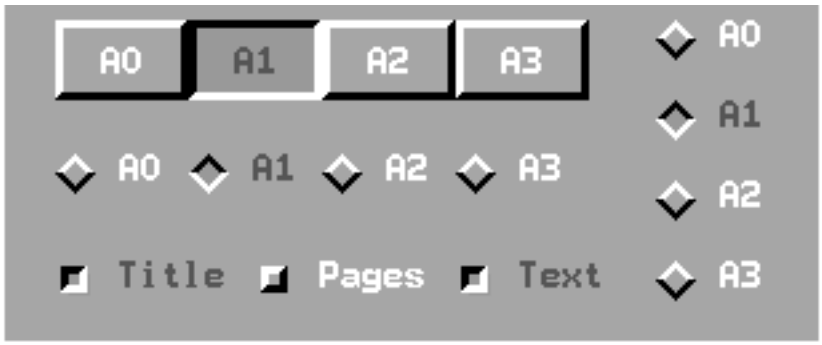
The STYLE attribute is one of three used in combination to display the QCHOICE item in different ways. The others are LAYOUT and MODE.

Please note: The multimode button operation, invoked by the MODE,MULTI attribute, converts the style of the diamond-shaped radio buttons to square buttons as for the MOTIF style of user interface. In multiselection mode, the modified definition to provide a horizontal group of multiselect buttons is as follows:

```
STYLE,TOGGLE,MODE,MULTI,
```

When STYLE is used with a QCHOICE item, its only valid argument is TOGGLE. By default no STYLE is defined. This attribute is applicable only to the QBUTTON and QCHOICE items.

Figure 4-10 Combinations of STYLE,TOGGLE, LAYOUT,V, & MODE,MULTI Attributes for QCHOICE Items

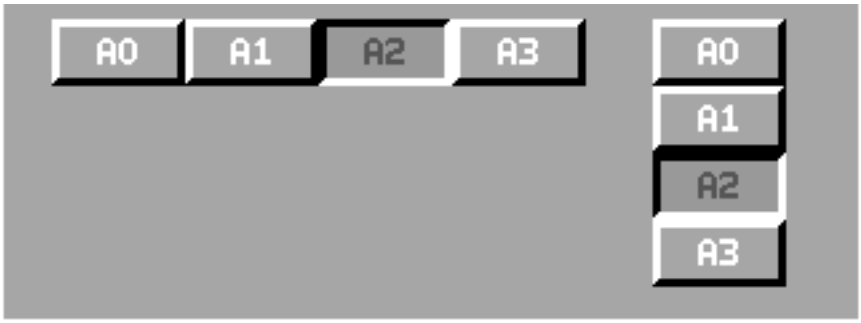


LAYOUT

The LAYOUT attribute is used with the QCHOICE type item only. It defines whether the list of options is displayed vertically or horizontally.

- H the options are shown as a horizontal row of buttons.
- V the options are shown as a vertical list of buttons.

Figure 4-11 `QCHOICE` **Items with** `LAYOUT,H` **&** `LAYOUT,V` **Attributes**



By default, the options are listed horizontally. To list options vertically, include the following in the attribute list:

```
LAYOUT,V,
```

The `LAYOUT` attribute can be used in conjunction with the `MODE` and the `STYLE` attributes to define the overall style and presentation of the list of options for a `QCHOICE` item.

The `WIDTH` and `HEIGHT` attributes define the size of an individual button. Thus, the number of buttons determines the overall width (horizontal buttons) or height (vertical buttons) of the listed options as a multiple of the relevant dimension.

When designing a `QCHOICE` item, check that this overall dimension does not exceed the limits of the window.

For the radio buttons presentation of `QCHOICE` items (using the `STYLE,TOGGLE` attribute), the `WIDTH` attribute defines the hot spot area (the selection area) for the button.

The default for this attribute is `LAYOUT,H`. It is applicable only to the `QCHOICE` item.

Please note: The `STYLE` and `LAYOUT` attributes used significantly alter the size and shape of the area occupied by the `QCHOICE` item. Take special care when designing the window layout that the item does not overlap other items or exceed the limits of the window.

ICON,DICON

Buttons can be displayed with icons instead of, or as well as, text. The `QWINDOW` item can also include a `ICON` attribute.

QWINDOW: The ICON attribute to the QWINDOW item specifies the name of the icon to be used when the window is closed. Each window in an application can use its own icon.

Please note: The icon type available is dependent on the window system. In some environments only the window system icons can be used. Refer to "Creating and Using Icons" on page 3-40 for more information.

QBUTTON: QBUTTON items can use only AccessWare icons. On Windows NT, .bmp formats are allowed. For QBUTTON items, the ICON and DICON attributes enable you to display an icon on the button instead of the text and to distinguish between an unpressed button (ICON) and a pressed button (DICON) by using two different icons.

If no DICON is defined, the same definition as ICON is used for a pressed button. Both ICON and DICON are applicable only to QBUTTON. However, the ICON attribute can be defined for the QWINDOW and QLIST items.

Example:

```
ICON,/usr/icons/p1,DICON,/usr/icons/down_p1,STATE,ICONIC,
```

Please note: For the button to be displayed as an icon, the STATE attribute must be set to ICONIC. If a LABEL attribute is also given, the label is displayed below the icon.

ICONLABEL

The ICONLABEL attribute argument is a text string that is displayed under the Window icon when it is closed.

Example:

```
ICONLABEL,My Application,
```

The label specified by ICONLABEL is used instead of any text string from the window system. The default is no label for an icon. This attribute is applicable to the QWINDOW item.

Attributes to Control the Color of Items

Items in an interface can be displayed in a choice of sixteen colors selected from a color palette. Each color in the color palette file is identified by an integer value in the range 0 to 15.

Refer to “Using Colors in AccessWare” on page 3-43 for a full description on how to define the color palette.

ITEMCOLOR,TEXTCOLOR

The `ITEMCOLOR` attribute defines the foreground or main color of each item. Depending on the item, this can be the color of the panel, the foreground color of a deselected button, or the foreground color of the text field.

The `TEXTCOLOR` attribute sets the color of any `LABEL` text displayed in the item.

Both these attributes take a single integer that represents the number of the color to be used from the color palette file.

Example:

```
ITEMCOLOR,2,TEXTCOLOR,4,
```

Default colors are set by AccessWare (`ITEMCOLOR,14,TEXTCOLOR,11,`), but these can be respecified through an environment variable at runtime. These attributes are applicable all items except `QWINDOW`, `QPOPUP`, and `QGROUP`

Please note: Item colors set up by environment variables for an individual interface define the standard colors used. However, the color attributes enable these standard colors to be changed for localized areas of the interface.

BACKCOLOR,SELECTCOLOR

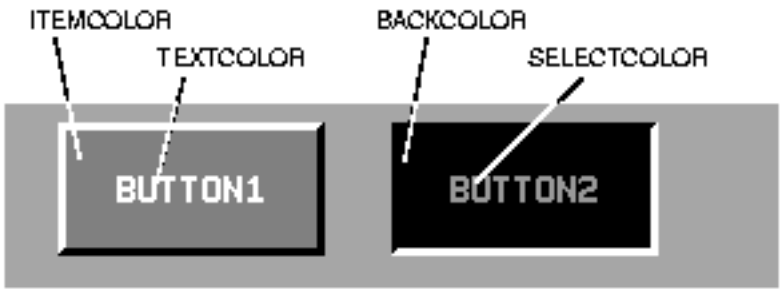
When a user selects an item (for example, by pressing a `QBUTTON` or choosing a `QCHOICE` option), you can specify item-selected colors which are the selected equivalents to `ITEMCOLOR` and `TEXTCOLOR` as follows:

Deselected Item	Equivalent Selected Item
ITEMCOLOR	BACKCOLOR
TEXTCOLOR	SELECTCOLOR

Thus, with the item selected, BACKCOLOR defines the button color and SELECTCOLOR defines the color of its label. Both these attributes take a single integer that represents the number of the color to be used from the color palette.

The following figure is an example of the different color attributes.

Figure 4-12 Examples Showing the Different Color Attributes



Example:

```
BACKCOLOR,2,SELECTCOLOR,4,
```

Default colors are set by AccessWare (BACKCOLOR,13,SELECTCOLOR,3,), these can be respecified through an environment variable at runtime. These attributes are applicable to all items except QWINDOW, QPOPUP and QGROUP.

Please note: SELECTCOLOR is not applicable to non-selectable items.

GREYOUTCOL

The GREYOUTCOL attribute enables you to define the color for graying-out items that are hidden or deactivated. It is useful when items are superimposed on a colored panel.

For example, if a button in a yellow panel is on a grey background, the standard aw_hide_item request repaints the button grey and not in the yellow of the panel. By specifying GREYOUTCOL as yellow, the button is hidden correctly.

The attribute takes a single integer that represents the number of the color to be used from the color palette.

Example:

```
GREYOUTCOL,5,
```

The default grey-out color is the color assigned the color number 0 in the color palette currently used by the application. The GREYOUTCOL attribute is applicable to all items except QWINDOW, QPOPUP, QGROUP, and QMENU.

Please note: Item colors set up by environment variables for an individual interface define the standard colors used. However, the color attributes enable these standard colors to be changed for localized areas of the interface.

Attributes to Assign Labels to the Item

A label displays a text string within an item. It is used to identify the purpose or use of the item, or identify what input or user action is expected.

LABEL

A text label can be associated with a range of different items. For most items, the LABEL attribute defines a text string that is always displayed in an item. For a QPOPUP item, the LABEL attribute instead identifies the pathname of the resource file where the new window invoked by QPOPUP is defined.

The following paragraphs describe how to define the LABEL attribute for each item to which it applies.

QWINDOW: The LABEL attribute defines the text that appears on the window header stripe. The single argument text string is centered on the header.

Example:
```
LABEL,Signon Window,
```

A label in the header stripe should explain the purpose of the window to the user.

QBUTTON: The LABEL attribute defines the text that appears on the button. The single argument text string is centered on the button.

Example:
```
LABEL,QUIT,
```

Please note: If an icon is also displayed on the button, by using a ICON attribute in conjunction with the Attribute STATE,ICONIC, the label is displayed below the icon. A label is not displayed when an icon is specified.

QTEXT,QMESSAGE,QPANEL: The LABEL attribute defines a short prompt that appears in the field. The single argument is the text string. This fixed text always

appears in the item and is used to identify the field and prompt user input. The following example defines the LABEL attribute for a text field.

Example:
```
. . . LABEL,Text: ,. . .
```

Figure 4-13 Example of a LABEL Attribute for a Text Field

The text is positioned in the top left hand corner for a QPANEL item. The text is left justified and vertically centered in the field for QTEXT and QMESSAGE items.

Please note: If a QTEXT item is defined as MODE,MULTI, the input text automatically wraps to the number of rows specified with the ROWS attribute. In this case, do not use the LABEL attribute.

QCHOICE: The QCHOICE item displays a list of buttons. The LABEL attribute defines the labels that appear on or beside the buttons for each option that appears in the list. Each option must be separated by the tilda character (~), and the list of options terminated with the keyword QQ.

Example:
```
LABEL,A0~A1~A2~A3~QQ,
```

The button labels and the terminating QQ are all separated by the character (~). This example is shown in the following figure.

Figure 4-14 QCHOICE Item Showing LABEL Attribute Options

QPOPUP: The LABEL attribute has a special function with a QPOPUP item. It defines the name of the resource file in which the window called by the QPOPUP item is defined. Its single argument gives the pathname of the resource file.

Example:
```
LABEL,/usr/files/window.res,
```

By default the LABEL attribute is not specified. It is applicable to all items except QLIST, QGROUP, and QMENU.

Attributes to Define the Field Contents

The field contents are defined with the following attributes.

LISTFILE

The argument to the LISTFILE attribute identifies a standard text file that provides a list for use with the item.

QLIST: For a QLIST item, the LISTFILE attribute identifies the name of the text file containing its list of options.

Example:
```
LISTFILE,/usr/abc/drawings_list,
```

The user can view the contents and, where items are suitably configured, can scroll left, right, up and down the list. Lines from the list can also be selected for further operations.

The mechanisms used with QLIST items for AccessWare interfaces are described in “Working with Lists” on page 3-27.

QMENU: For the QMENU item, the LISTFILE attribute identifies the name of a Menu Item Definition File, which contains the list of menu options and defines the actions invoked by each option.

The format of the Menu Item Definition File and information on how to construct menus for use with AccessWare interfaces is detailed in “Working with Menus” on page 3-35.

Example:
```
LISTFILE,/usr/abc/mymenufile,
```

The file /usr/abc/mymenufile holds the menu definition.

QGROUP: The QGROUP item defines a set of items controlled by a common action. The LISTFILE attribute identifies the name of a text file containing the list of items forming the group.

Example:
```
LISTFILE,/usr/abc/itemgroupfile,
```

The file `/usr/abc/itemgroupfile` lists the items in the group definition.

By default the `LISTFILE` attribute is not specified. It is applicable only to the `QLIST`, `QGROUP`, and `QMENU` items.

ITEMFILE

This attribute is used with menu related items such as the `QMENU` item. Its function is similar to the `LISTFILE` attribute. See "LISTFILE" on page 4-37 for details.

Example:
```
ITEMFILE,/usr/abc/menu_list,
```

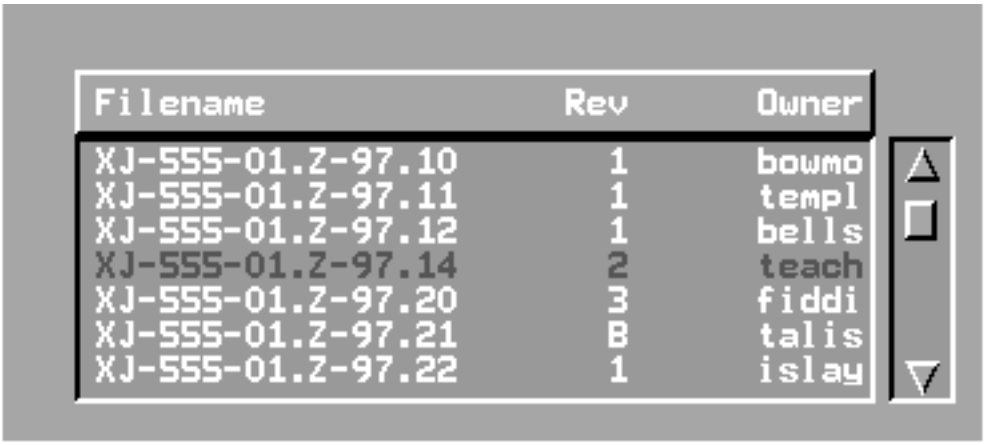
LISTMENU

The `LISTMENU` attribute defines a menu that is related to a `QLIST` item. The menu is obtained from the list when the right mouse button is clicked. It then acts as any other `QMENU` item.

For example, in a list of file names, `LISTMENU` can define a set of potential actions for any file name the user selects from the list as illustrated in the following figure. The argument of `LISTMENU` is the `QID` of a `QMENU` item.

Example:
```
LISTMENU,menuname,
```

Figure 4-15 `QLIST` **Item with a** `LISTMENU` **Attribute**



For a sequence of operations, the user selects a file from the list, brings up the menu, and selects an action from the supplied menu options.

By default the LABEL attribute is not specified. It is applicable to all items except QLIST, QGROUP, and QMENU.

By default the LISTMENU attribute is not specified. It is applicable only to the QLIST item.

MAXVALUE, MINVALUE

The MAXVALUE and MINVALUE attributes are used with the QSLIDER item to define the maximum and minimum values of a slide bar.

Example:

```
QSLIDER
QID,xsliderr,ETCH,4,XPOS,0,YAFTER,0,WIDTH,130,HEIGHT,25, ->
LAYOUT,H,LABEL,0~100~QQ,VALUE,10,ACTION,1001,MINVALUE,0, ->
MAXVALUE,100,VISIBLE,1,QEND
```

Attribute to Supply Initial Defaults to an Item

Use these attributes to supply initial defaults to an item.

VALUE

A value can be associated with some items. Depending upon the item to which it relates, the VALUE attribute has the following functions:

- a parameter value supplied to the application for a selected QBUTTON
- the initial default selection for a QCHOICE item
- the default user input text for a QTEXT item
- the number of rows or characters to scroll for a QPANEL item configured as a horizontal scrollbar (ACTION,HS) or vertical scrollbar (ACTION,VS).

When the VALUE attribute is included in the attribute list, it effectively sets an initial default value for an item. This parameter can be subsequently changed by actions occurring during the execution of the application.

If the argument to the VALUE attribute is specified as an environment variable, it is expanded at runtime and the item value displayed accordingly. This enables initial field values to be set from the application at runtime.

The effect of the VALUE attribute depends upon the item with which it is used as described in the following paragraphs.

QBUTTON: The VALUE attribute provides a parameter value that is associated with the button. This value can be supplied as an argument to the action invoked by the button.

Example:

```
VALUE,10,
```

This feature is used when a set of buttons all invoke the same ACTION. The VALUE attribute can then supply a different value for the action according to the button selected.

QCHOICE: The argument to the VALUE attribute supplies the initially selected option in the choice list. The options are numbered from 1 to the number of options. A value of 0 denotes that no option is selected.

Example:

```
VALUE,3,
```

Figure 4-16 Default Presentation of a QCHOICE Item

In this example, the list of items is shown initially displayed with the third option selected. Any subsequent action that changes the VALUE attribute of the QCHOICE item changes the default selected option.

QPANEL: The VALUE attribute applies only to a QPANEL item configured as a scrollbar. Its argument must be an integer number and specifies the number of rows to scroll for a QPANEL vertical scrollbar, or the number of columns to scroll for a QPANEL horizontal scrollbar.

Example:

```
VALUE,3,ACTION,VS,
```

The preceding example defines the QPANEL item as a vertical scrollbar (using the ACTION,VS attribute) that scrolls by three rows each time the scroll button is pressed.

QTEXT: The VALUE attribute for a QTEXT item supplies the default text string that appears in the input field. If a text prompt defined by the LABEL attribute is also specified for a QTEXT item, then the default input text defined for the VALUE attribute follows on from this prompt.

Example:

```
VALUE,root,
```

Figure 4-17 QTEXT **Item with** VALUE **Attribute**

An application action can subsequently change this value.

By default the VALUE attribute has the value 0 for the QCHOICE item. There is no default for this attribute in the case of the QBUTTON, QPANEL, and QTEXT items. This attribute is applicable to the QBUTTON, QPANEL, QCHOICE, and QTEXT items.

Please note: The VALUE attribute provides the initial default when the application is started. The VALUE of an item can be reset subsequently using the procedure call aw_set_item_value, or by the SV action.

Attributes to Assign Special Characteristics

Use the following attributes to assign special characteristics to items.

STATE

The STATE attribute sets a state for the item. It is followed by another keyword that sets a type of state.The permitted keywords and the way the STATE attribute is used is depends on the item type. The options available are described in the following paragraphs.

QTEXT: The STATE attribute sets the user access state for a QTEXT item. Its argument can be one of the following keywords:

- READONLY — Locks the item to prevent the user entering or updating the value or contents of the item.
- SECURE — Echoes all user input characters back to the user as asterisks. This feature is useful for fields where the user enters a password or pin number.

The READONLY state may subsequently be changed to allow the user to MODIFY the item by using procedure calls. By default, neither is set.

QBUTTON: For a QBUTTON item, the STATE attribute specifies whether or not the button is displayed with an icon.

- ICONIC — Specifies icon display of the button.

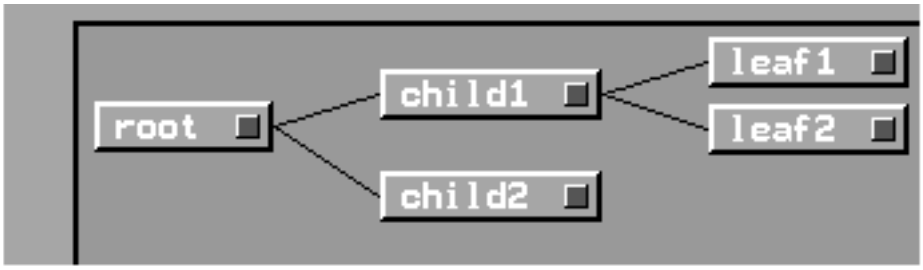
Even though an icon may be specified for the button, it is not displayed with an icon unless the STATE,ICONIC attribute is included in the QBUTTON definition.

QLIST: The STATE attribute used with a QLIST item may take a QLIST attribute to control the way in which the list is presented and subsequent operations that may be performed upon it.

- TEXT — Specifies text format for display of list contents.
- TREE — Specifies that the list is displayed as a tree.

  The TREE state for a QLIST item implies that the list contents are appropriately formatted for display as a tree as illustrated in the following figure. More information is given in"Displaying Trees in AccessWare" on page 3-32.

Figure 4-18 QLIST **Item with a** STATE,TREE **Attribute**




```
Defaults:None (lists are displayed in text form if no STATE
attribute is given.
Applicable to:QTEXT, QBUTTON, QLIST
```

There are no defaults for the STATE attribute. Lists are displayed in text form if the STATE attribute is not defined. It is used only to provide these special modifying functions. This attribute is applicable only to the QTEXT, QBUTTON and QLIST items.

MODE

The MODE attribute is used with QCHOICE and QLIST items to set the item selection mode and with QTEXT items to control the input text mode.

QTEXT: For a QTEXT item, the argument of the MODE attribute controls how the characters input by the user are echoed and displayed in the field and passed to the application. The keyword arguments that can be used are:

- UPPER — Translates all input text to upper case before it is passed to the application and echoed in the input field.
- LOWER — Translates all input text to lower case before it is passed to the application and echoed in the input field.
- REAL — Provides a validation function by accepting only real decimal numbers and rejecting all other input characters. The validated characters are passed to the application and echoed in the input field.
- INTEGER — Provides a validation function by accepting only integers and rejecting all other input characters. The validated characters are passed to the application and echoed in the input field.
- MULTI — Specifies that the text field can be used for multiline input with intelligent word-wrapping.

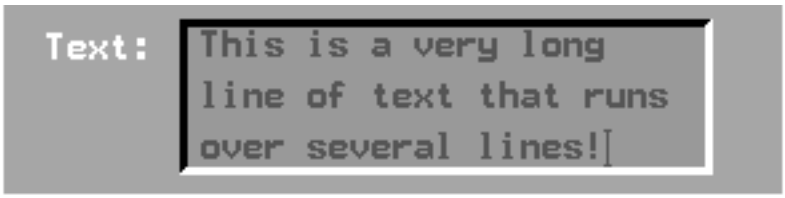
Example:
```
MODE,UPPER,
```

Unless the MODE attribute is specified, the user input stream is unaffected.

Please note: The form of text echoed in the display is that sent to the application, that is, case converted where specified.

MODE,MULTI sets the display mode for the text field to multiline, but is limited to the number of rows of text defined by the ROWS attribute.

Figure 4-19 QTEXT **Item with a** MODE,MULTI **Attribute**



QCHOICE,QLIST: When the MODE attribute is used with the QLIST and QCHOICE items, it defines whether the user can select single options or multiple options from the list. The valid option is:

- MULTI — Specifies multiselection mode, where any number of options can be selected at one time.

Example:
```
MODE,MULTI,
```

Unless MODE,MULTI is specified, the default single-selection mode applies, that is, only one item can be selected at a time, such that selecting an option automatically deselects the previous selection.

In this mode, word-wrap is initiated and at the extremity of the field intelligently wraps the text into the available number of rows, that is, if a word exceeds the space to the end of a line, it is displayed on the next line. This is illustrated in Figure 4-19. Consequently, the text field sometimes contains fewer characters than might be expected from ROWS multiplied by COLS. The text is still delivered to the application as a single line text string.

Please note: If the text field height is instead defined using the HEIGHT attribute, the field is then only a single line field and displays only a single line of text truncated by the width of the field.

By default text is displayed in mixed case for a QTEXT item. The default MODE for QCHOICE and QLIST items is single mode.

Please note: Only one MODE attribute may be specified. multiline text permits no other MODE options.

DEFAULT

This attribute allows you to specify an item as the default action taken when the RETURN key is pressed. It is used with the QBUTTON item.

Example:

```
QBUTTON
XPOS,90,YAFTER,5,WIDTH,100,HEIGHT,22,LABEL,$OK_LABEL,->
ACTION,7310,MOVEY,100,MOVEX,100,VALUE,1,MODE,DEFAULT,QEND
```

FOCUS

This attribute allows you to specify an item to be active when the window is opened. Set the value of this attribute to 1 for a particular item to activate it when the window is opened. This attribute can be used with any item.

Please note: When multiple items in a window's resource file have a `FOCUS` value set to 1, only the last item with this setting is activated when the window is opened.

Examples:

```
QPANEL
QID,CA_HEADER,XPOS,1,YAFTER,1,WIDTH,$MAIN_TBAR_WIDTH,->
HEIGHT,30,STRETCHX,100,DEPTH,1,ETCH,1,FOCUS,0,QEND

QTEXT
QID,TRANSFER_NAME,XAFTER,10,YOFFSET,0,WIDTH,300,MAXCOLS,80,->
HEIGHT,22,INVERT,1,DEPTH,1,PROMPT,$CA_ASSYT_MSG18,->
STRETCHX,100,FOCUS,1,QEND
```

OVERVIEW

This attribute allows you to display a scaled down view of a tree without its node labels. The actions in the original tree such as node selections are automatically reflected in the overview tree. This attribute is used with the `QLIST` item and is used as follows:

`...,OVERVIEW,`tree_name`,...`

Here, tree_name is the name of the tree for which you want an overview.

Example:

```
QLIST
QID,CA_OVERLIST,FONT,$CA_SMALL_FONT,VALUE,15,INVERT,1,XPOS,1,->
YPOS,1,HEIGHT,395,WIDTH,$CA_OVERTREE_TWIDTH,->
LISTFILE,$CA_MAINTREE,DEPTH,2,STATE,TREE,ACTION,6905,->
ITEMCOLOR,$CA_NODE_COLOR,SELECTCOLOR,$CA_NODE_SEL_COLOR,->
BACKCOLOR,$CA_NODE_BACK_COLOR,STRETCHX,100,STRETCHY,100,->
OVERVIEW,CA_TREELIST,QEND
```

INTERVAL

This attribute allows you to specify the interval by which a slide bar increases or decreases. This attribute is used with the `QSLIDER` item.

Example:

```
QSLIDER
QID,pvs_move_slider,ETCH,4,XPOS,50,YAFTER,50,WIDTH,130,->
HEIGHT,25,LAYOUT,H,LABEL,Limits,VALUE,10,ACTION,60615,->
MINVALUE,0,MAXVALUE,100,INTERVAL,5,VISIBLE,1,QEND
```

Miscellaneous Attributes for Items

The following are miscellaneous attributes for items.

FONT

The `FONT` attribute specifies a text font to use in the interface. The items to which the `FONT` attribute is assigned determines the extent to which it applies (that is, the items/windows covered by the font definition).

The three valid levels at which a font can be defined are:

- Within the `QWINDOW` definition of the main resource file.

  All items in the entire interface use the specified font, except where a font is separately defined for an individual item or individual window.

- Within any other `QWINDOW` definition.

  The font applies to all items only within the window concerned, except where a font is separately defined for an item.

- Within any other valid item definition.

  The font applies to only the item concerned, overriding any font definitions for parent items.

You can define fonts for all item types, and use different fonts within a single window. By default the font is the window system font.

Example:

```
FONT,/sys/fonts/timesxxxx,
```

The FONT attribute is applicable to all items except the QPOPUP and QGROUP items.

Please note: Specify fonts with a fixed width, as the width of the font is used to calculate the character column space. Consequently, proportional spacing fonts produce unpredictable results.

RULES

The RULES attribute is used in the QEXPRESSION item to create a boolean expression of two or more control states. Operators supported for this expression object are && (and) and || (or).

Example:

```
QEXPRESSION
QID,SYSADMINORPROJADMIN,VALUE,0,RULES,->
CA_STATE_SYSADMIN_SIGNON || CA_STATE_PROJADMIN_SIGNON,QEND
```

UDATA

The UDATA attribute is used to define user data using the QMENUITEM.

For example, in the old format user data for the menu item Reports > Parts_List (where Reports is the menu and Parts_List is an item in it) would be defined in the Menu Item Definition File as follows:

```
Parts_list  6500    PERLREP ALL     $CA_REPORT_DIR/partlist.pl
```

As there are only QMENUITEM QIDs in a Menu Item Definition File in the new format menus, in order to implement the same menus using QMENUITEM, UDATA is introduced. Therefore, in the new format for Reports > Parts_List the QMENUITEM definition would be as follows:

```
QMENUITEM
QID,CA_MENUID_REP_PARTS_LIST,VISIBLEDEP,CA_EXPR_STD_OR_SAP_OR_CM_
OR_CAMU,LABEL,$CA_MENUID_REP_PARTS_LIST_LBL,ACTION,6500,->
UDATA,$CA_MENUID_REP_PARTS_LIST_LBL      6500      PERLREP ALL
$CA_REPORT_DIR/partlist.pl,QEND
```

Please note:

- The complete line for the Parts_List menu now appears as a UDATA definition of QMENUITEM for Parts_List.
- The ACTION field contains the action number for the menu item which will also be present as part of the UDATA string.
- The user can also use this attribute to place any data, which can be retrieved as a string using the function aw_get_udata ( ).

Attributes to Invoke the Help Facilities

The following attributes provide help facilities for the user:

- PROMPT — Specifies a text string, which is displayed in the QPANEL item set up as a Prompt field by ACTION,PR.
- HELPFILE — Specifies a text file containing help text, which is displayed in a Help window.
- HELPWIN — Identifies the QID of the Help window used to display a helpfile defined by the HELPFILE attribute
- TIP — Specify quick information about the button action in a tooltip.

PROMPT

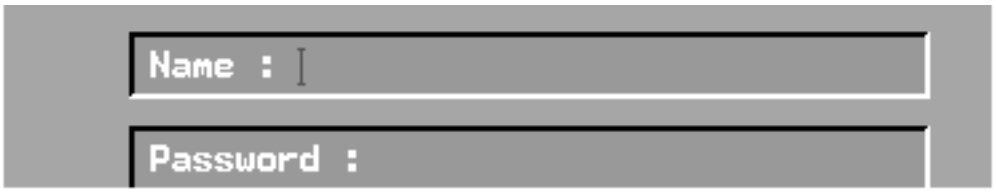
The PROMPT attributes provides a help facility for the user in the form of a short message. You can define a simple prompt string for any item.

Using the PROMPT attribute, context-sensitive help can be defined for any item in a window and displayed in a suitably labelled QPANEL item as seen in the following figure. The QPANEL item chosen for the prompt area is configured to accept these prompts by assigning an ACTION,PR attribute to it.

The string is displayed in the prompt area whenever the mouse cursor enters the area encompassed by the item, thus providing a simple level of help to assist the user for all items.

Example:
```
PROMPT,Enter your name,
```

Figure 4-20 `QTEXT` **Item with a** `PROMPT` **Attribute**



By default no prompt is specified. This attribute is applicable to all items except `QWINDOW`, `QPOPUP`, `QMENU`, and `QGROUP`.

HELPFILE

The `HELPFILE` attribute identifies the name of a text file containing detailed information about the item selected.

The helpfile is displayed in the Help window defined for the interface, which is in turn specified by the `HELPWIN` attribute defined for the main window of the application. This window is invoked by the `ACTION,HELP` attribute whenever the help facility is selected for any item.

Example:

```
HELPFILE,/usr/help/help_helpfile,
```

By default no help is provided. This attribute is applicable to all items except `QWINDOW`, `QPOPUP`, `QMENU`, and `QGROUP`

HELPWIN

The `HELPWIN` attribute identifies the name of the Help window and is assigned to the main window of the application. The Help window applies to the whole interface.

Example:
```
HELPWIN,helpwin_QID,
```

The HELPWIN attribute is an attribute to the QWINDOW item of the main window resource file. Its argument specifies the QID of a QPOPUP item that represents the Help window. The QPOPUP item in turn identifies the resource file that defines the Help window.

The Help window resource file contains all the items necessary to provide a comprehensive help facility, allowing the user to view and scroll through the help files. It must include one QLIST item, which is used to display the help text, but includes other items as necessary, such as QPANEL scrollbars.

When Help is invoked for an item, the application copies the contents of the item's helpfile to the QLIST item's listfile (that is, the generic file name defined by the ITEMFILE attribute of QLIST). It is then displayed in the Help window.

By default no help is provided. This attribute is applicable to the QWINDOW item (main window only).

Please note: A full description for setting up a comprehensive help facility in an AccessWare application is given in "Providing Context Sensitive Help" on page 3-46.

TIP

The TIP attribute is a helpful suggestion about how to use a particular button. This attribute is used only for the QBUTTON item.

Example:
```
TIP,Opens a file,
```

Attributes That Affect the Whole Application

The following attributes set characteristics for the interface. They can only be used with the QWINDOW item for the main window definition.

RESIZE

This attribute allows you to specify a direction in which to resize a window.

Example:
```
...,RESIZE,1,...
```

The following options are available with this function:

- `0` — Resizing is disabled for the window
- `1` — Resizes the window both horizontally and vertically
- `2` — Resizes the window vertically
- `3` — Resizes the window horizontally
- `4` — Resizing is disabled for the window. However, the window can be minimized to an icon and opened independent of the state of the application's main window.

Please note: When `RESIZE` is set to 4, the `AW_WINDOW_STATE` environment variable must be set to 1.

SPLASH

This attribute allows you to specify the file name of an image to use as a splash screen.

Example:
```
...,SPLASH,/usr1/image/splash.jpg,...
```

NOBANNER

By default, AccessWare displays a banner page whenever any application is run. This page is written to the screen before the initial window is displayed. The banner displays information such as current software revision, and other generic information.

Display of the banner is suppressed with the attribute keyword `NOBANNER`, which must be entered as an attribute to the `QWINDOW` item of the main resource file.

Example:
```
NOBANNER,
```

By default a Banner is displayed.

NOSTRIPE

The stripe is the host window system border at the top of a window and appears on each AccessWare window. Using Unix, the stripe appears as a header bar. On the PC, the standard tool stripe is available.

This stripe makes it easier for the user to manipulate the window using host window system functions such as move, size, or close.

The stripe is suppressed with the keyword NOSTRIPE, which must be entered as an attribute to the QWINDOW item of the main resource file. This function does not apply to Motif style systems.

Example:
```
NOSTRIPE,
```

Please note: The Quit option has been removed from the stripe menu to prevent the user quitting the application before AccessWare or the application has had time to tidy up properly.

By default a stripe is displayed.

PALETTE

The PALETTE attribute defines a color palette file for the interface. It must be defined as a QWINDOW attribute in the main resource file.

Example:
```
PALETTE,palettes/my_palette,
```

A default color palette file is provided with the AccessWare installation, but you can define your own color palette to suit the requirements of the application. See "Using Colors in AccessWare" on page 3-43.

A default palette is at `palettes/access.pal`.

GLOBVARS

The GLOBVARS attribute specifies the file containing a list of environment variables to be used with the application.

Example:

```
GLOBVARS,filename,
```

The form of the list in the specified GLOBVARS file is as follows:

```
environment_variable = value
```

The maximum number of global variables that can be defined in a GLOBVARS file is 512.

Please note:

- There are no apostrophes or similar special characters enclosing the item values in the GLOBVARS file.
- The values specified in the GLOBVARS file take precedence over any Environment Variables specified outside the application.

Global Variables are used as substitution values for attribute arguments or parts of arguments in the list of attributes for the items. Refer to “Setting Environment Variables” on page 2-4 for more information.

By default the GLOBVARS attribute is not defined.

Please note: A PC based AccessWare interface can only use Global Variables defined in the GLOBVARS file.

ACTION Attribute Reference

AccessWare provides a range of standard actions to reduce, or in some cases eliminate, the amount of coding and script writing that might otherwise be required.

An action is associated with an AccessWare item by the ACTION attribute. Although actions are associated with buttons, they can be linked to any item in the window.

For all items other than QWINDOW, the action is performed when the user clicks the mouse button on the item concerned.

An ACTION specified for a QWINDOW item is performed when the window is initially created, and again as the application exits. This provides a facility for setting initial item values and for tidying up on exit.

Invoking ACTIONs in AccessWare

An ACTION can be associated with any item in a window. Actions form the link between the user interface and the application. They are initiated as a result of some activity in the interface, such as the clicking of a button, and result in an event or action taking place in the application or elsewhere in the interface.

AccessWare provides the following ways to generate an action:

- Through a recognized standard built-in AccessWare action.

  These provide a means to perform various standard actions such as hiding or showing items, scrolling lists, and closing or exiting the application. They enable many applications to be developed without the need to write or call C programs.

- Through a call to an external script or an application program.

  The Script Action Interface: This enables an item to invoke an operating system script permitting more sophisticated actions to be undertaken without the need for C programming knowledge.

  The Programmatic Interface: This provides an interface to the user's application programs from an AccessWare user interface.

This chapter describes the ACTIONs that provide standard built-in functions. These can be used directly by items in the window to perform actions in the AccessWare interface, and they can be used in conjunction with the Script Interface.

The Script Interface is described in Chapter 5, “AccessWare Script Interface”, and the Programmatic Interface is covered in Chapter 6, “AccessWare Programmatic Interface”.

Form of the ACTION Attribute

If an action is associated with an AccessWare item, the attribute list must include the `ACTION` attribute.

The `ACTION` attribute takes the following forms:

- `ACTION,integer_number,`

  This invokes a piece of code through the Programmatic Interface.

- `ACTION,keyword,argument_list,`

  This form initiates one of the built-in functions described in the following paragraphs.

The `ACTION` attribute can be included anywhere in the attribute list for an item.

Please note: An item can initiate a series of actions, but they must be listed in a `command file`. Refer to “CF — Action Command File” on page 4-57 for more information.

ACTION Attribute Summary

The `ACTION` attributes can be categorized in functional groups as follows:

Application Interface Actions	
Integer	Identify an element of application code
The Action Command File	
`CF`	Specify command file list of AccessWare actions
Actions that Operate on Windows	
`CL`	Close a window
`EX`	Exit from a window
Actions that Modify the State of an Item	
`GO`	Deactivate (grey-out) an item
`GI`	Activate (grey-in) an item
`HI`	Hide an item
`SH`	Show an item

Actions that Set Values for Items

SV	Set a value for an item

Actions that Denote Special fields

ME	Configure a QPANEL for an application message
PR	Configure a QPANEL for all items Prompt text

Actions that Invoke Menus and Popups

MU	Invoke a popup menu
POP	Invoke a popup window
POPREL	Position popup window relative to an item

Actions to Call Scripts

SY	Call to execute system command or run a script
SCR	SY call plus arguments
UL	SCR call plus update QLIST item after command
MACRO	Defines a macro containing Perl script statements.

Actions that Issue CADDS 5i Commands

CAD	Send a line to CADDS
CAD1	Send a line to CADDS without carriage return

Special Actions for QPANELs Associated with QLISTs

HEAD	Configure a QPANEL as scrolling header bar
VS	Configure a QPANEL as vertical scrollbar
HS	Configure a QPANEL as horizontal scrollbar

Actions to Toggle the Help Facility

HELP	Toggle system help facility on and off

Application Interface Coding Actions

The Programmatic Interface enables users to invoke their own programs from actions in an AccessWare interface. It includes a suite of C procedures that enable the user's own programs to access items in the interface and manipulate their contents.

To use the C Programmatic Interface, the ACTION must simply specify an integer value. This number identifies a section of C code to be executed when the user clicks the item concerned.

Example:

```
ACTION,1000,
```

The mechanism used to action the code is as follows:

- AccessWare calls the procedure `aw_user_exit` and passes the `ACTION` number, an item identifier, the item value, and the mouse state to it as arguments.
- The `aw_user_exit` procedure comprises a switch statement that contains a list of case statements, each identifying by a label the appropriate section of code for the action required. The statements following the specified label are executed.

The `aw_user_exit` procedure is called each time the mouse is clicked in an item, or when a window is created or removed.

The application programmer must write, compile, and link his or her own `aw_user_exit` procedure to create his or her application.

The structure of the `aw_user_exit` procedure to build an application using the Programmatic Interface is described in Chapter 6, "AccessWare Programmatic Interface".

Please note: Action numbers `0-500` are reserved for AccessWare use. Consequently, all action numbers specified by the application programmer must be `>500.`

The Action Command File

In the Action Command File, use the following attributes.

CF — Action Command File

With the CF attribute, you specify a text file that contains a list of AccessWare actions.

Synopsis:
```
ACTION,CF,action_filename,
```

The `CF` attribute is useful when one item must initiate more than one action. In addition the action file can be redefined dynamically at runtime.

For example, to run a script that initiates printing from a Print button and then greys-out (deactivates) the button, you could specify an `ACTION` on the Print `QBUTTON` item as follows:

```
... ACTION,CF,/usr/files/my_actions,...
```

This initiates the print function. Create a script file `/usr/files/my_actions` containing the following:

```
SCR,/usr/scripts/my_script,file_name~QQ,
GO,button_qid,
```

This deactivates the button.

Any of the AccessWare standard actions can be included within the file, it can also include `SY`, `SCR`, or `UL` actions to run scripts. In addition, the file can include other `CF` actions nested to any level.

ACTIONS That Operate on Windows

AccessWare provides a number of actions that operate on windows.

CL — Close Window

This action closes the current window, reducing it to an icon. The icon used is defined by the `ICON` attribute in the `QWINDOW` definition.

Synopsis:
```
ACTION,CL,
```

EX — Exit Window

The `EXIT` action removes the current window from the screen.

Synopsis:
```
ACTION,EX,
```

If the window is the last window displayed in the interface, the application exits.

Please note: The application does not ask for confirmation from the user. If confirmation is required, it should be an action programmed into the application.

Actions That Modify the State of an Item

The following action attributes enable items to be hidden or shown, and activated or deactivated.

GO — Grey Out Item

This action deactivates (greys-out) an item that has been:

- initially set to the active state by the ACTIVE attribute, or
- activated by a previous GI action.

Identify the item by its QID keyword.

Synopsis:
```
ACTION,GO,item_qid,
```

The item is then unavailable to the user although it may still be visible in the window, as determined by the setting of the GREYOUTCOL attribute.

This action is related to the ACTIVE attribute of the item, which sets the initial state of the item as activated or deactivated when the application starts. The ACTION,GO attribute overwrites the value set by the ACTIVE attribute with ‘0’ in order to deactivate the item.

GI — Grey In Item

This action activates (greys-in) an item that has been:

- initially set to the inactive state by the ACTIVE attribute, or
- deactivated by a previous GO action.

Identify the item by its QID keyword.

Synopsis:
```
ACTION,GI,item_qid,
```

This action is related to the ACTIVE attribute of the item, which sets the initial state of the item as active or deactive when the application starts. The ACTION,GI attribute overwrites the value set by the ACTIVE attribute with ‘1’ in order to activate the item.

HI — Hide Item

This action hides an item that has been:

- initially set to visible by the VISIBLE attribute, or
- made visible by a previous SH show action.

Identify the item by its QID keyword.

Synopsis:
```
ACTION,HI,item_qid,
```

This action is related to the VISIBLE attribute of the item, which sets the initial state of the item as visible or invisible when the application starts. The ACTION,HI attribute overwrites the value set by the VISIBLE attribute with '0' in order to hide the item.

SH — Show Item

This action displays an item that has been:

- initially set to hidden by the VISIBLE attribute, or
- hidden by a previous HI hide action.

Identify the item by its QID keyword.

Synopsis:
```
ACTION,SH,item_qid,
```

This action is related to the VISIBLE attribute of the item, which sets the initial state of the item as visible or invisible when the application starts. The ACTION,SH attribute overwrites the value set by the VISIBLE attribute with '1' in order to show the item.

Actions That Set Values for Items

The following action enables a new value to be set for an item.

SV — Set Value

The SV action sets or resets a series of values for an item. It overwrites any existing values set initially by the VALUE attribute or any previous SV action.

The argument of SV is the name of a file, which contains a number of rows defining the values of the items specified.

Synopsis:
```
SV,filename,
```

The SV file contains one or more definitions, each on a separate line, each definition in the file takes the form:

```
qid = value
```

where qid is the QID of the item concerned and value is the content of the VALUE attribute to be set or reset.

In the following example, a file sets two item values as follows:

- a QTEXT item named user_name to contain My Name
- a QCHOICE item named mychoice to select option 4

The file contains the following entries:

```
user_name = My Name
mychoice = 4
```

The VALUE attribute holds different information depending upon the item to which it relates, as follows:

- QBUTTON — A parameter value supplied to the application for a selected button.
- QCHOICE — The default selection from the available options.
- QTEXT — The default user input text.
- QLIST — The line number of the selected row.
- QPANEL — The number of rows or characters to scroll when the panel is configured as a scrollbar (using ACTION,HS or ACTION,VS).

Actions That Denote Special Fields

The following actions configure a QPANEL item as field to display special messages:

- ME —Displays any application message
- PR — Displays any prompt text defined by the PROMPT attribute (that is, to provide context-sensitive help)

ME — Message Action

The ME action configures a QPANEL item to display a message from the application.

Synopsis:
ACTION,ME,

The application message is displayed in the QPANEL with the ACTION,ME attribute in the currently active window.

The message delivered to the ACTION,ME panel is read from the file defined by the Environment variable AW_MESSAGE_FILE. It is read after the execution of every ACTION.

A message can be sent to the message file using the procedure aw_message. The message string is the argument to this function call. The file can also be created by a script.

For further details on using application messages refer to Chapter 5, “AccessWare Script Interface” and Chapter 6, “AccessWare Programmatic Interface” and *AccessWare Function Reference*.

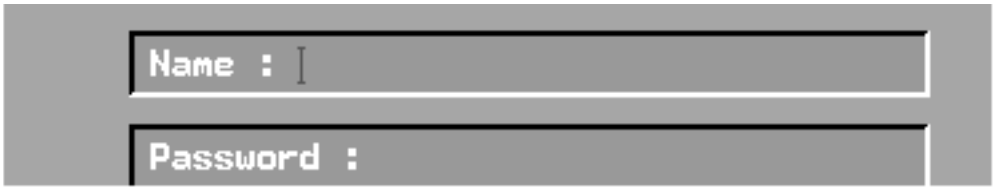
PR — Prompt Action

The PR action configures a QPANEL item to display Prompt text. The Prompt text is displayed as the mouse cursor moves into the space encompassed by the item.

Synopsis:
ACTION,PR,

The Prompt text is defined for an item using the PROMPT attribute, and its argument specifies the Prompt text for that item.

Figure 4-21 QTEXT **Item with** PROMPT **Attribute &** QPANEL **Item with** ACTION,PR **Attribute**



When the user moves the mouse cursor into the QTEXT item field, any prompt text defined for it is displayed in the QPANEL item that has the ACTION,PR attribute defined in the same window.

Actions That Invoke Menus and Popups

The following ACTION attributes are applied to items that must invoke a menu or a popup window when they are selected.

MU — Invoke a Menu

The MU action invokes a menu that is displayed when the user clicks the left mouse button on an item. It is used with either a QTEXT, QMENUITEM or QMESSAGE item.

Identify the relevant menu by the QID name of the QMENU item in the resource file that references the required Menu Item Definition File.

Synopsis:
```
ACTION,MU,menu_name_qid,
```

This action can be applied to any QTEXT field. The ACTION,MU action can also be used, with a series of QMESSAGE items, to create a set of drop-down menus along a menu bar.

POP — Invoke a Popup Window

The POP action invokes a popup window that is displayed when the user clicks the left mouse button on the current item.

Identify the relevant window by the QID name of the QPOPUP item in the resource file that references the new window resource file.

Synopsis:
```
ACTION,POP,pop_up_window_qid,
```

The POP action can be used to invoke a window from any simple window element except QWINDOW, QPOPUP, and QGROUP.

Popup windows may be used either as simple popups to present another set of features of the application related to a current activity, or as a new window to which the application passes overall control.

POPREL — Invoke and Position a Popup Window

The POPREL action invokes a popup window and defines the position at which it is displayed relative to the current item.

Identify the relevant window and related item by specifying the QID names given to the associated QPOPUP and the relevant item in the resource file.

Synopsis:
```
ACTION,POPREL,pop_up_window_qid,item_qid,xoffset,yoffset,
```

xoffset and yoffset define the position of the window in pixels relative to the top left corner of the item concerned.

This ACTION ensures that the popup window is always displayed adjacent to the item to which it relates even if the window containing the item is moved. It overrides any positioning attributes given for QPOPUP in the resource file definition.

Actions to Call Scripts

The following actions execute an operating system command or script, and provide a link to the application.

SY — System Call

The SY action makes an external system call. It either executes an operating system command, or runs a script.

Following the SY keyword, specify the name of the system command or the name of the script file.

Synopsis:
```
ACTION,SY,system_command,
```

Such a command can be executed by putting it in a script and executing the script, or by using an alias or some other similar technique supported by the operating system.

Please note: Since arguments in the attribute list are delimited by commas the system command line cannot include any commas. However, complete commands with arguments delimited by spaces can be sent to the system (but a script containing such a command could be initiated instead).

SCR — Run Script

The SCR action provides the link to the Script Interface. It makes an external system call, similar to the SY action, but in addition passes a set of arguments to it.

Following the SCR keyword specify the name of the script followed by a list of arguments.

Synopsis:
```
ACTION,SCR,script_name,argument_list,
```

The argument list can contain one or more script arguments. Separate each argument with the character: tilda (~) and end the list with QQ.

The arguments may include VALUES associated with other items in the interface. To pass an item value, include the QID of the item in the argument list.

For example, to pass the values associated with a QCHOICE item with a QID of mychoice and a QTEXT item with a QID of file name, the ACTION definition is:

```
ACTION,SCR,script_file,mychoice~filename~QQ,
```

Please note: The ~QQ is required even if a single argument is given.

UL — Update List

The UL action is an extension of the SCR action. The ACTION,UL attribute performs the same action as the ACTION,SCR attribute, but then updates the QLIST item specified following the argument list.

Synopsis:
```
ACTION,UL,command_name,argument_list~QQ,list_qid,
```

The ACTION,UL refreshes the list contents of the QLIST list_qid with the latest contents of its ITEMFILE file, which presumably has been updated during the system call.

For example, the ACTION attribute for a Delete button that deletes a file selected from a list is as follows:

```
ACTION,UL,my_delete_script,filename~QQ,mylist_qid,
```

Please note: You can pass the currently selected line in the QLIST as an argument to this attribute by including its QID in the arguments list. The value of a QLIST item is the row number selected.

MACRO — Define a Macro

This action causes the mouse cursor to become busy. Upon completion of the macro the mouse cursor returns to normal. Currently only Perl macros are allowed. This helps in avoiding the use of `aw_cursor_busy()` calls in the application to change the mouse cursor mode while executing Perl scripts using `ACTION`,SCR. The following is an example of how this action is used.

Example:

```
QBUTTON
XAFTER,15,YPOS,50,WIDTH,80,HEIGHT,25,LABEL,MyMacro,VISIBLE,1,TIP,
Macro Executes,ACTION,MACRO,/users/scripts/macro-perl-script,QEND
```

The following is an example of a Perl script macro:

```
#!/usr/CVperl/bin/perl

use Aw;

Aw'start_macro();
print "Macro Execution Started. Type a Message\n";
$temp = <STDIN>;
print "$temp\n";
print "Macro Execution Ends\n";
Aw'exit_macro();
```

Actions That Issue CADDS Commands

The following ACTION attributes enable a command to be sent to the CADDS window.

CAD — Send CADDS Command

The `CAD` action sends a line to the CADDS text window. The command includes a terminating carriage return. CADDS then executes the command.

Synopsis:
`ACTION,CAD,cadds_command_line,`

Please note: Since the CADDS command line is delimited in the attribute list by commas, it may not include commas.

CAD1 — Send CADDS Command without CR

The CAD1 action performs the same function as the CAD action but omits the terminating carriage return.

Synopsis:
```
ACTION,CAD1,cadds_command_line,
```

Please note: Since the CADDS command line is delimited in the attribute list by commas, it may not include commas.

Special Actions to Enable Scrolling of QLISTs

These special actions are applied to QPANELs that provide scrollbars or scrolling header bars for QLISTs.

Setting up scrollbars and header panels for use with QLIST items is described in "Working with Lists" on page 3-27.

HEAD — Header Bar Action

This action configures a QPANEL item as a scrolling header bar for a specified QLIST item. When the list is scrolled, the header bar contents (the titles for the list fields) also scroll in vertical alignment with the list fields.

Synopsis:
```
ACTION,HEAD,list_qid,
```

HS — Horizontal Scrollbar

This action configures a QPANEL item as a horizontal scrollbar for a specified list, and displays it with an elevator and scroll buttons at each end as seen in the following figure.

Synopsis:
```
ACTION,HS,list_qid,
```

Figure 4-22 QPANEL **Item (Scrollbar) with** ACTION,HS **Attribute**

The action is applied to the QPANEL item that is required to horizontally scroll the specified list. The QPANEL may also include a VALUE attribute specifying the number of columns by which to scroll.

VS — Vertical Scrollbar

This action configures a QPANEL item as a vertical scrollbar for the associated list.

Synopsis:
```
ACTION,VS,list_qid,
```

The action is applied to the QPANEL item that is required to vertically scroll the specified list. The QPANEL may also include a VALUE attribute specifying the number of lines by which to scroll.

Scrolling Actions for QBUTTONs

AccessWare also provides a set of standard actions to assist with scrolling of lists using QBUTTON items. The shape of the buttons can also be changed to triangular arrowheads for this purpose by assigning the STYLE attribute to these QBUTTON items. Refer to “STYLE” on page 4-29 for more information.

Actions to Toggle the Help Facility

Use these actions to invoke a help facility.

HELP — Toggle Help Action

This action is included as an attribute to an item that switches on the system help facility. This could be a menu option or a button.

The HELP action toggles on and off the function that causes any selected item to provide help instead of its normal action. The resultant functions of the application are:

1. A Help window is invoked, which is identified by the HELPWIN attribute in the main QWINDOW resource file as a QPOPUP item.
2. The contents of the selected item’s help text file is copied into the generic help text file of the QLIST item defined for the Help window.
3. The contents of the help text file are displayed in the Help window.

Please note: The selected item’s help text file is identified by its HELPFILE attribute, and the generic help text file of the QLIST is identified by its ITEMFILE attribute.

Synopsis:
```
ACTION,HELP,
```

The help facility toggle action can be initiated, by a Help button or by one of the options provided on a dropdown menu. The `HELP` action is then included either in the definition of the `QBUTTON`, or in the Menu Item Definition File respectively.

Within an application, there may be a number of toggle items enabling Help to be switched on and off in various places.

Please note: When the `HELP` action is toggled on, the cursor changes to a question mark, indicating that it is in Help mode and will display the help file for any selected item.

Chapter 5

AccessWare Script Interface

This chapter describes how to use the Script Interface to execute external scripts. It allows you to interface AccessWare `ACTIONs` to run script files, access data, and update the user interface.

- Overview
- Using the Script Interface
- Script Interface Examples

Overview

The AccessWare Script Interface provides ACTIONs to execute a text file containing operating system commands. In UNIX this file is referred to as a script. An interface running on a PC executes the specified batch file.

Please note: If the Script Interface is used in a PC based application, set up the batch file to run in the window environment.

In the resource file, the ACTION attribute attached to an item, identifies the type of action performed when the item is selected. The text file contains a sequence of commands appropriate to the operating system under which AccessWare is running.

The ACTION attributes that interface user scripts are as follows:

- SCR: The ACTION,SCR attribute calls a script file passing a set of arguments to it. The format is as follows:

  ACTION,SCR,command_name,argument_list~QQ

- UL: The ACTION,UL attribute also calls a script file in the same way as the SCR attribute, but on return updates the contents of a specified list. The format is as follows:

  ACTION,UL,command_name,argument_list~QQ,list_qid

Please note:

- Include the QID of the QLIST whose contents are to be updated when defining the ACTION,UL attribute.
- For both ACTIONS the command_name is the name of the script to be executed.
- Separate each argument in the list of arguments by a tilda character (~) and the list must be terminated by ~QQ.

You can also use the ACTION,SY attribute to run a command which is a script. but no arguments can be passed to it. For more information on ACTION attribute see “ACTION Attribute Reference” on page 4-54.

Using the Script Interface

The most common requirements for interfacing with an application are as follows:

- Passing arguments from the interface to a script

  Arguments that you need to pass include values in text fields, values associated with buttons, and toggle settings on QCHOICE items.
- Passing an item selected from a list to a script
- Updating a text field or a toggle button setting
- Updating the contents of a list

Pass the value of an item in the interface to the script as an argument by including its QID in the argument list. Update it through an SV action. The SV action specifies a text file containing a list of QID names and the new values to be assigned to them. This action is implemented using an AccessWare command file, which includes the SCR action followed by an SV action.

Displaying Messages from a Script

Using the ME action you can display messages in the current window message area.

During execution of a script action, include commands to send the required message to the message file. The AW_MESSAGE_FILE environment variable defines the message file. AccessWare automatically reads this file and, on completing the action, displays the text in the item with the ACTION,ME attribute in the currently active window.

Script Interface Examples

This section provides examples of applying the Script Interface as part of building a user interface.

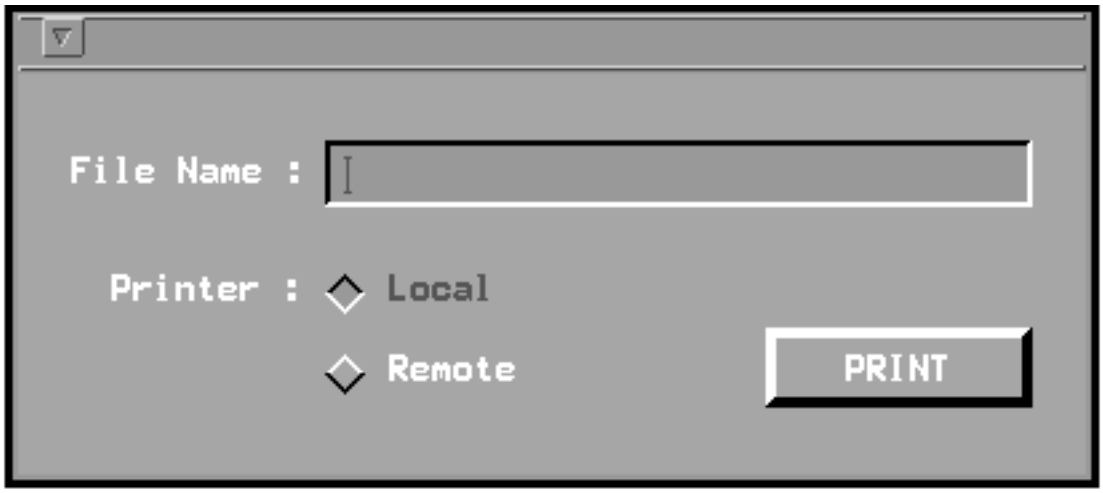
Passing Arguments to a Script File

The following is an example of a resource file that provides a printing facility in a user interface. It enables the user to specify a file name, select a printer, and to initiate the print operation by means of the PRINT button.

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,150,LABEL,Print File,QEND
# File Name input text field
QMESSAGE
XPOS,20,YPOS,31,COLS,10,ROWS,1,LABEL,File Name:,DEPTH,0,QEND
QTEXT
QID,filename,XPOS,115,YOFFSET,-6,WIDTH,265,HEIGHT,25, ->
INVERT,1,QEND
# Printer fields
QMESSAGE
XPOS,35,YPOS,75,COLS,9,ROWS,1,LABEL,Printer:,DEPTH,0,QEND
QCHOICE
QID,printer,XPOS,115,YOFFSET,0,WIDTH,150,HEIGHT,30, ->
STYLE,TOGGLE,LAYOUT,V,LABEL,Local~Remote~QQ,VALUE,1,QEND
# The PRINT Button
QBUTTON
XPOS,280,YPOS,95,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,PRINT, ->
ACTION,SCR,print_scr,filename~printer~QQ,QEND
# ============================================================
QATTREND
```

The panel created by this resource file is shown in the following figure:

Figure 5-1 Example of a Simple File Print Panel



In this example, the item definitions include a QTEXT item for the user to enter a filename and a QCHOICE item to select a printer.

The print function is invoked by the QBUTTON labelled PRINT which has an ACTION it to run a script in the operating system.

Executing the Script File

In the preceding example, the QBUTTON has SCR action that runs a script in the text file print_scr. The arguments in the script are the values of the QTEXT and QCHOICE items.

The arguments are as follows:

- first argument — Supplies the filename that is the value of the QTEXT item, passed by referencing the QID of the item
- second argument - Supplies the toggle button setting. This is the value of the QCHOICE item again referenced by specifying the QID of that item

On a UNIX system, the text file `print_scr` is as follows:

```
#!/bin/csh
# Script arguments are  1) Name of file to print
#                       2) Printer choice selected
#
# Set printer name according to value of QCHOICE
# Local printer is 'lw1', remote printer is 'lw2'
if ($argv[2] == 1 ) then
    set prname=lw1
else
    set prname=lw2
endif
#
# Issue the print statement
lpr -P$prname $argv[1]
#
# End
```

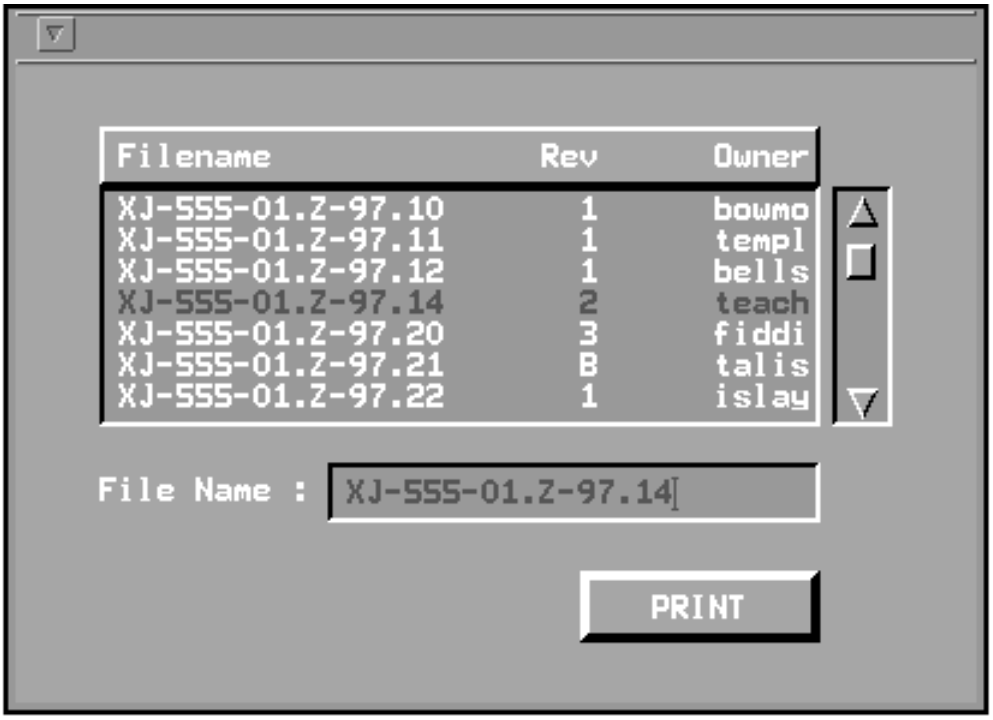
Once the script has been executed control is passed back to the AccessWare application.

Please note: The terminology used in these examples is that associated with UNIX systems. On a PC for example, a text file containing a set of operating system commands is referred to as a batch file.

Updating a Text Field

This example enables the user to select an item from a list, which is then used to update the contents of an input text field. The following figure shows the panel created by the resource file.

Figure 5-2 Example of a List and Print Panel



The resource file for this example is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,265,LABEL,Print Window,QEND
# File List Field
QLIST
QID,filelist,XPOS,35,YPOS,50,WIDTH,300,HEIGHT,100,INVERT,1, ->
LISTFILE,mylist,ACTION,CF,myactions,QEND
# Vertical Scrollbar for List
QPANEL
XAFTER,5,YOFFSET,0,HEIGHT,100,WIDTH,25,INVERT,1, ->
ACTION,VS,filelist,VALUE,2,QEND
# List Header Bar
QPANEL
XPOS,35,YPOS,25,HEIGHT,25,WIDTH,300,ACTION,HEAD,filelist,QEND
# File Name input text field
QMESSAGE
XPOS,35,YPOS,171,COLS,10,ROWS,1,DEPTH,0,LABEL,File Name:,QEND
QTEXT
QID,filename,XPOS,130,YOFFSET,-6,WIDTH,205,HEIGHT,25,INVERT,1,QEND
# PRINT Button
QBUTTON
XPOS,235,YPOS,210,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,PRINT, ->
ACTION,SCR,print2_scr,filename~QQ,QEND
# ============================================================
QATTREND
```

In this example the QLIST that has an associated ACTION is run using an AccessWare command file. The ACTION,CF attribute identifies the text file myactions, which contains a list of AccessWare actions.

Description of the ACTION Attribute

The CF action file myactions, invoked by the QLIST item when a line in the listing is selected, is as follows:

```
SCR,get_list_scr,filelist~QQ
SV,/tmp/newvalues
```

The actions that the file myactions performs are:

1. Execute the SCR action to set up a setvalues text file:
2. Run the script get_list_scr and pass to it the value of the QLIST with the QID of filelist.

   The value of a QLIST is the line number of a selected row
3. Create a new file /tmp/newvalues

   The script fetches the text from the selected row of the listfile and uses it to create the file /tmp/newvalues in the form qidname = value.
4. Execute the SV action to update the value of the text field

   The file /tmp/newvalues contains the line: filename=name where name is the name of the file extracted from the listing.

On a UNIX system, the text file get_list_scr is as follows:

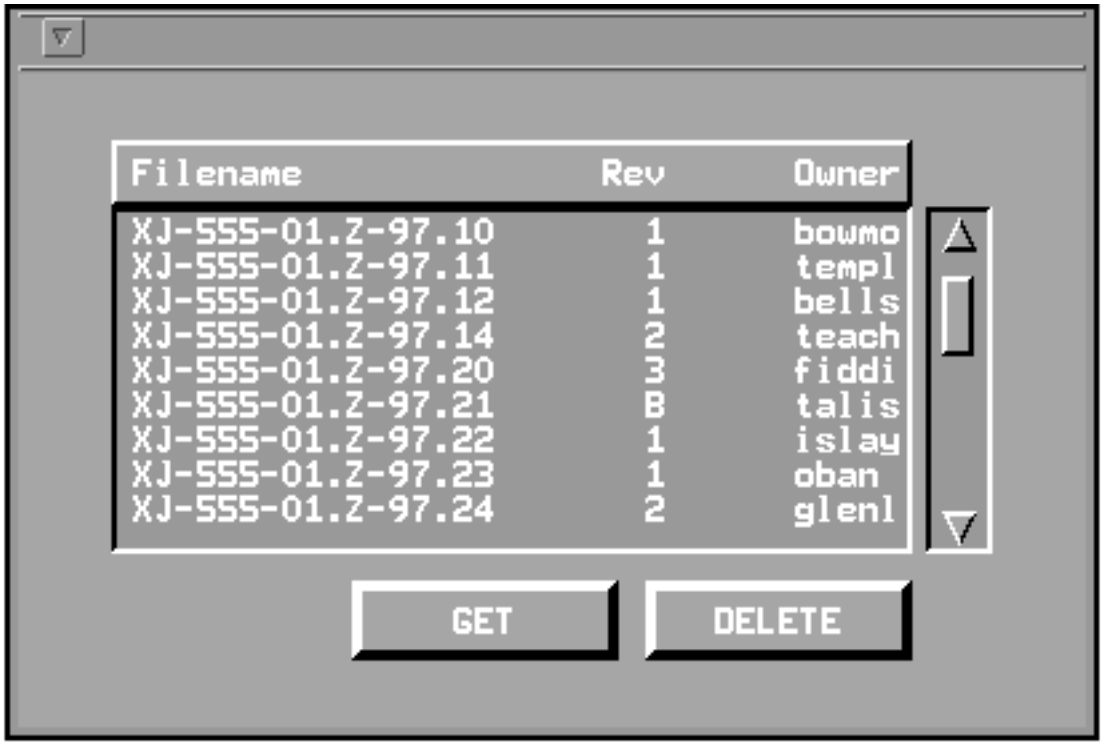
```
awk '{if(NR == lineno)
       {
           printf "filename=%s\n",$1
       }
}' lineno=$1 mylist > /tmp/newvalues
```

The value in the text field could now be used as an argument to the action of the PRINT button, similar to the previous example.

Updating List Contents

This example shows a panel that enables a user to select an entry from a list, perform an action on it, and then update the list. The panel for this example is shown in the following figure.

Figure 5-3 Example of a List Requiring an Update Action



The resource file to create the preceding panel is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,245,LABEL,List of Files,QEND
# File List Field
QLIST
QID,filelist,XPOS,35,YPOS,50,WIDTH,300,HEIGHT,130,INVERT,1, ->
LISTFILE,mylist,ACTION,CF,myactions,QEND
# Vertical Scrollbar for List
QPANEL
XAFTER,5,YOFFSET,0,HEIGHT,130,WIDTH,25,INVERT,1, ->
ACTION,VS,filelist,VALUE,2,QEND
# List Header Bar
QPANEL
XPOS,35,YPOS,25,HEIGHT,25,WIDTH,300,ACTION,HEAD,filelist,QEND
# GET Button
QBUTTON
XPOS,125,YPOS,190,WIDTH,100,HEIGHT,30,DEPTH,4, ->
LABEL,GET,ACTION,SCR,get_scr,filelist~QQ,QEND
# DELETE Button
QBUTTON
XPOS,235,YPOS,190,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,DELETE, ->
ACTION,UL,delete_scr,filelist~QQ,filelist,QEND
# ============================================================
QATTREND
```

Description of the ACTION Attribute

In the preceding example, for the QBUTTON labelled DELETE an ACTION,UL attribute is included to update the QLIST once the script actions are completed.

```
QBUTTON
XPOS,235,YPOS,190,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,DELETE, ->
ACTION,UL,delete_scr,filelist~QQ,filelist,QEND
```

The resultant actions are as follows:

1. Executes a script named delete_scr, passing to it the argument filelist. This is the row number of the file selected from the list.
2. Creates a temporary file containing the command to delete the selected file. This temporary file is then executed to delete the file.
3. Generates a new directory listing and sends it to the list contents file mylist, overwriting the previous contents.
4. Refreshes the listing displayed in the QLIST with the latest version of the list contents file.

On a UNIX system, the text file delete_scr is as follows:

```
#!/bin/sh
# Remove any existing temp file
rm -f tmp_scr
#
# Get the selected row from 'mylist' and send it
# to a temporary file with a delete command
awk '{ if( NR == lineno)
      {
            printf "rm %s\n",$1
      }
}' lineno=$1 mylist > tmp_scr
chmod +x tmp_scr
#
# Execute the temporary file to delete the file
tmp_scr
#
#  Re-generate the list contents file
ls -C1 > mylist
```

Most actions that involve selecting and manipulating items in a list should end with an update of the list. The ACTION,UL attribute gives the user an updated list of contents after performing the other actions.

Chapter 6

AccessWare Programmatic Interface

This chapter describes how to use the AccessWare Programmatic Interface that enables users to invoke their own programs from actions in an AccessWare interface.

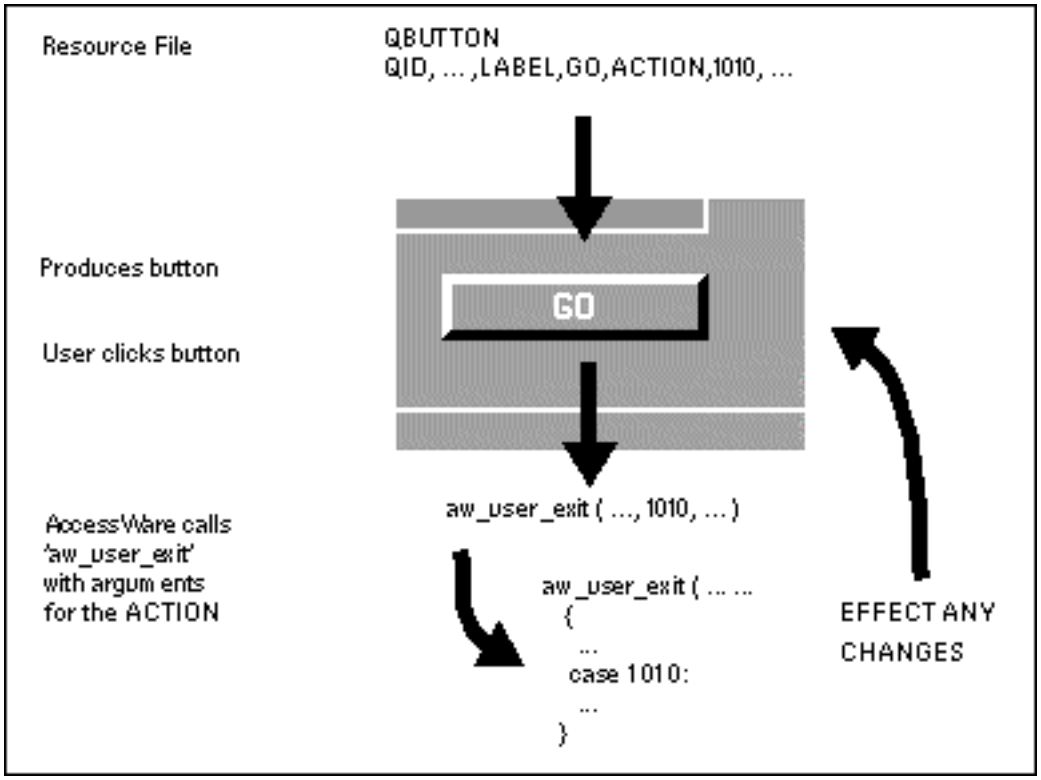
- Overview
- Structure of the Application Code
- Compiling and Linking Your Program
- The AccessWare Procedure Library
- AccessWare Programming Examples

Overview

AccessWare Programmatic ACTIONs can be associated with any item in a window. They form the link between the user interface and the application code. They are initiated as a result of some activity in the interface, such as the clicking of a button, and result in a piece of code being executed in the application.

The Programmatic Interface includes a suite of C procedure calls that allow a user's own programs to access items in an interface and manipulate their contents. Thus, window items can be manipulated in conjunction with events in the application, for example, buttons can be disabled when their actions are inappropriate. The following diagram shows the elements of the Programmatic Interface.

Figure 6-1 The Programmatic Interface



Applying an ACTION

If an ACTION is associated with an AccessWare item, the attribute list must include the ACTION attribute. To invoke a piece of code through the Programmatic Interface the ACTION attribute must simply specify an integer value:

- ACTION, integer_number

  This number identifies a section of C code to be executed when the user clicks the item concerned.

The following mechanism executes the code:

- AccessWare calls the procedure aw_user_exit and passes the ACTION number and other data as arguments.
- The aw_user_exit procedure includes a switch statement that contains a list of case statements, each identifying by a label the appropriate section of code for the action required. The statements following the specified label are executed.

The aw_user_exit procedure is called each time the mouse is clicked on an item or when a window is created or removed.

Using Libraries

When using AccessWare procedure calls and/or CMOM APIs you need to link to the following CMOM libraires:

- Windows NT:
  - %EPD_HOME%\lib\optmdll.lib
  - %EPD_HOME%\lib\optmlib.lib
- Unix:

  $EPD_HOME/lib/libcvoptmsg.so

Structure of the Application Code

The application programmer must write a `main` procedure to initialize the application environment and the `aw_user_exit` procedure (or other exit procedures as required) to supply the actions required when items are selected.

The main Procedure

The structure of the procedure `main` follows:

```
#include "access.h"

main (argc, argv)

int argc;
char ** argv;

{
      /* initialization of application environment */

      /* Pass control to accessware */

      aw_init_windows ( "main resource file path name");

      /* Tidy up ready for exit */

      exit ();
}
```

Programmers must define their own `main` procedure to set up the application environment. The `main` procedure must include a call to the procedure `aw_init_windows`, to initialize and pass control to AccessWare.

Once control has been passed to AccessWare, all application code can only be accessed using the procedure call `aw_user_exit`.

The aw_user_exit Procedure

The `aw_user_exit` procedure is the standard mechanism for accessing any application code from AccessWare. The procedure comprises a switch statement which passes control to one of a series of case statements, each of which identifies, by an integer label, a section of code for the action required.

The format of a typical `aw_user_exit` procedure follows:

```
aw_user_exit(...., action_number,...)

switch(action_number)
```

```
{
      /* Action QBUTTON, QID x, QWINDOW, win_signon */

      case  1000
            {
            /* Get other item values */
            /* Perform X/Exit action */
            }

      /* Action QBUTTON item, QID go, QWINDOW, win_signon */

      case  1010
            {
            /* Get other item values */
            /* Perform the GO action */
            }
      ...
}
```

The application programmer must write, compile, and link the `aw_user_exit` procedure in this form to control the application.

Action numbers in the range 0 - 500 are reserved for use by AccessWare. Consequently, all action numbers specified by the application programmer must be greater than 500.

To assist with Modular programming, you can create a number of `user_exit` procedures that AccessWare can call. The procedure is explained later in this chapter.

Please note: Allocate different ranges of numbers to different modules in your application. Also, allocate action numbers in multiples of say ten, so that other actions can be added later in the same ranges as related actions.

The procedure call to `aw_user_exit` is as follows:

```
aw_user_exit (region, user_action, value, mouse)
```

The arguments have the following types and descriptions:

Type	Description
region integer	an identifier of the item that called the action.
user_action integer	the `ACTION` number from the resource file.
value integer	the value associated with the VALUE attribute of the item invoking the action.

Type	Description
mouse integer	either the state of the mouse button or other window condition. mouse button states: = MOUSE_DOWN: button has been pushed down = MOUSE_UP: button has just been released other window conditions: = WINENTER: window has just been created = WINEXIT: window has just been exited = WINREPAINT: window has been repainted = WINSIGNAL: a signal has been received = WINLF: a Line Feed has been entered = WINTAB: the TAB key has been entered = WINKEYASCII: an ASCII key has been hit = WINRETURN: the `Return` key has been hit

The `aw_user_exit` procedure is called twice for every full click of the mouse button. Thus, separate actions can be performed independently on both down and up mouse button movements. However, only a single action is required, so your code must check the state of the mouse button and perform the action only for the appropriate state.

Supplied include Files

Two AccessWare include files are supplied with your release, which you may need when writing your `aw_user_exit` procedure. They define a range of parameters, structures, and procedure types.

These files are provided in the AccessWare `/include` directory and are called `access.h` and `awfunc.h`. Always include `access.h` in all C program files that make calls to any AccessWare procedure:

```
#include "access.h"
```

The `access.h` file includes definitions to assist with checking the state of the mouse and creating and removing windows.

Two important parameters in the file are:

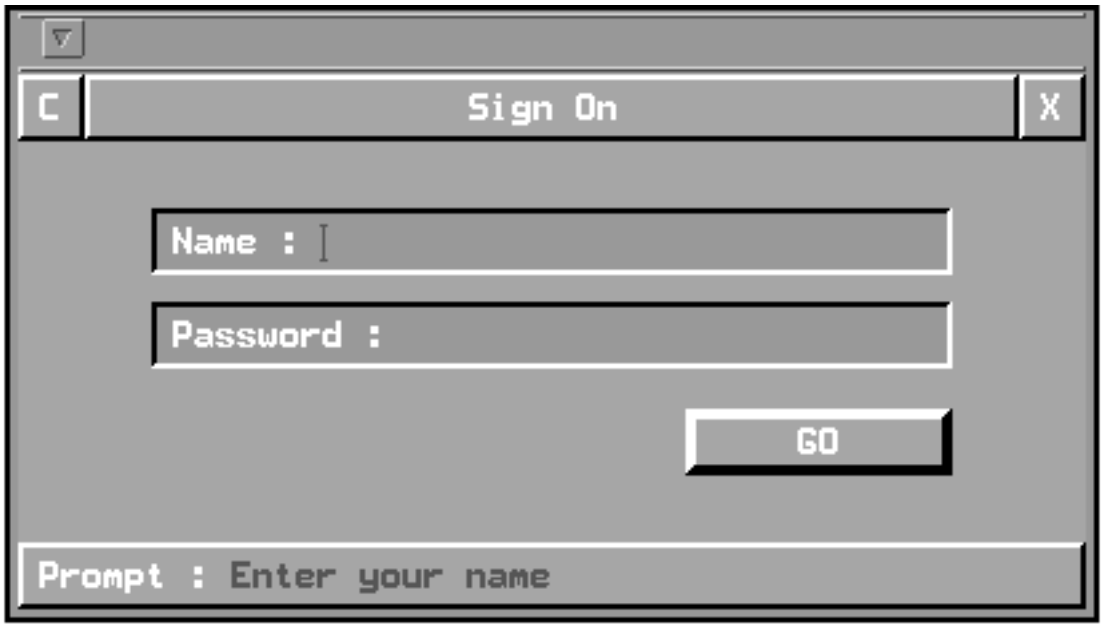
```
#define MOUSE_DOWN 0
#define MOUSE_UP 1
```

Please note: Reference the defined variables when checking the state of the mouse or the window.

Sample aw_user_exit Procedure

The example Sign On panel from Chapter 1, “Introduction to AccessWare” is shown in the following figure.

Figure 6-2 The Simple Sign On Panel example



The following code extract is from a file containing the aw_user_exit procedure for the preceding example:

```
#include ”access.h”

aw_user_exit (reg, user_action, value, mouse)

int reg, user_action, value, mouse;
{
 char   user_name[255], user_pw[255];

 switch (user_action)
 {
  /* Signon button action from signon window */
  case 1000:

    /* Check mouse button state */
    if ( mouse == MOUSE_UP )
    {
```

```
        /* Get the entered user name */
        strcpy ( user_name, aw_get_item_value ("name_qid"));

        /* Get the entered user password */
        strcpy ( user_pw, aw_get_item_value ( "pw_qid") );

        /* Check name and password */
        ...

        /* Hide the signon window */
        aw_hide_item ( "mainwin" );

        /* Show the next window */
        aw_show_item ( "win_pick" );
      }
    break;

    /* Another item action */
    case 1010:
      ...
  }
}
```

This code extract uses several AccessWare supplied procedures, specifically calls to procedures: `aw_get_item_value`,`aw_show_item`,and `aw_hide_item`.

A library of procedures is provided with AccessWare to access and manipulate items in the user interface. A list of these procedures is explained in the section "The AccessWare Procedure Library" on page 6-11. A full list of these procedures is given in the *AccessWare Function Reference* manual which is a companion volume to this manual.

Modularising the User Exit Code

For many applications, the `aw_user_exit` procedure can very quickly become quite large if the action code is directly added to each case statement. This makes with editing, compiling, and linking the programs difficult.

Consequently, you should put the code that performs the actions into separate procedures.

Implement modular programming techniques as follows:

- Put into `aw_user_exit` only the code that checks the state of the mouse and then calls an appropriate procedure. Put all the code that processes the action into that procedure.

Alternatively,

- Define additional `user_exit` procedures to AccessWare. Add the name of your new `user_exit` procedure into the AccessWare include file `awfunc.h`, together with the range of action numbers for which this new `user_exit` should be called, as follows:

```
extern int aw_user_exit () ;
extern int aw-my_exit ()   ;

AWUserExit aw_callbacks [] =
{
    { "userexit"    ,aw_user_exit , 100  , 2000  },
    { "myexit"      ,aw_my_exit   , 2000 , 10000 },
    { NULL          ,NULL         , -1   , -1    },
};
```

Please note: This procedure must have the same arguments as the default `aw_user_exit`. If you use this method, you must include `awfunc.h` as well as `access.h` in your main procedure.

Compiling and Linking Your Program

The `main` and `aw_user_exit` procedures must be compiled and linked to form an executable application. The compilation should be able to access the `access.h` include file and the link should refer to the AccessWare library.

The `access.h` include file is in the directory $EPD_HOME`/include`, where $EPD_HOME is the directory where the AccessWare software is installed.

Please note: To compile and link on a PC platform use the appropriate makefile in $EPD_HOME`/obj/....`

Compiling

Use the following commands to compile your source code files on a UNIX platform:

```
cc -c -I$ACCESSDIR/include main.c
cc -c -I$ACCESSDIR/include aw_user_exit.c
```

Linking

The AccessWare libraries are located in the $EPD_HOME`/lib` directory. There may be several libraries in this directory for the various window systems supported.

- If your application is to run under X windows or Motif, use the library `libcvawx11.a`.

To create an executable application named `test.exe` using the X/Motif libraries, use the following to link your object files:

```
cc -o text.exe main.o aw_user_exit.o ->
   $EPD_HOME/lib/libcvawx11.a
```

Sample makefiles are provided on your release tape under the directory $EPD_HOME`/obj`.

The AccessWare Procedure Library

A suite of procedures that allow access to and manipulation of your user interface are supplied as part of AccessWare. These procedures enable you to get values from and put values into an item defined in your resource file. They also enable you to alter the visual appearance of items.

For example, you can remove items from the display and scroll lists to specified rows and columns. A list of these procedure calls and their arguments is given later in this chapter.

Some of these procedures use parameters defined in the include file `access.h`, described in the previous section.

Item Referencing

Most of the procedure calls require an input argument which specifies the name of the item that you want to access. The item is always referenced by the `QID` allocated to it in the resource file. Consider the following example:

```
QTEXT
QID,user_name,XPOS, .....
```

If the resource file contains a text field, as shown above, the contents of the field are obtained via a procedure call using the item name, for example, the `QID` for `QTEXT` is the value user_name.

The `QID` of the `QPOPUP` item which references the resource file of a window is used in procedure calls that refer to that window. Consider the following example:

```
QPOPUP
QID,next_window, LABEL,next_win.res,VISIBLE,0,QEND
```

If the resource file contains the reference `next_win.res`, as shown above, the `QID` that is passed to a procedure call to reference the window defined by `next_win.res` is next_window.

To reference the main window, i.e. the first window that is displayed when the application is initiated and which does not have an associated `QPOPUP`, use the identifier `mainwin`.

In this manual, any reference to item names or `QIDs` refers to the identifier given by the `QID` attribute.

Summary of Procedure Calls

This section provides a summary of procedure calls grouped in categories. Use it to ascertain which is the most likely call you need for a requirement. A full description of all procedure calls is given in the manual *AccessWare Function Reference.*

The procedure calls are divided into categories as follows:

- Initiating and Exiting AccessWare
- Retrieving Item Values and Attributes
- Setting Item Values and Attributes
- Updating Item Appearance
- Procedures Associated with Lists
- Procedures Associated with Tree Nodes
- Miscellaneous Procedures

Initiating and Exiting AccessWare

Procedure Call	Description
int aw_init_windows (resource_file)	Pass control to the AccessWare environment.
int aw_exit_windows (status)	Abort the application where a serious error occurs.

Retrieving Item Values and Attributes

Procedure Call	Description
int aw_get_item_position (qid, x, y)	Get the position of an item in pixels.
int aw_get_item_xposition (qid)	Return the value of the x location of an item.
int aw_get_item_yposition (qid)	Return the value of the y location of an item.
int aw_get_item_dimension (qid, width, height)	Get the size of an item in pixels.
int aw_get_item_width (qid)	Get the width of an item in pixels.
int aw_get_item_height (qid)	Get the height of an item in pixels.
int aw_get_item_cols (qid)	Return the width of an item defined by the COLS attribute.
int aw_get_item_rows (qid)	Return the height of an item as defined by the ROWS attribute.
int aw_get_window_position (qid, x, y)	Get the position of a window in pixels.
int aw_get_window_dimension (qid, width, height)	Get the size of a window in pixels.
int aw_get_window_width (qid)	Return the width of a window.
int aw_get_window_height (qid)	Return the height of a window.
int aw_get_item_depth (qid)	Return the depth of an item.
int aw_get_item_invert (qid)	Return the invert status of an item.
int aw_get_item_etch (qid)	Return the etch setting of an item.
int aw_get_item_itemcolor (qid)	Return the item foreground color of an item.
int aw_get_item_backcolor (qid)	Return the background color of an item.
int aw_get_item_textcolor (qid)	Return the text color of an item.
int aw_get_item_greycolor (qid)	Return the greyout color of an item.
char *aw_get_item_label (qid)	Get the current label string of an item.
char *aw_get_item_icon (qid)	Return the name of the icon defined for an item.
char *aw_get_item_dicon (qid)	Return the name of the selected icon of an item.
char *aw_get_item_font (qid)	Return the name of the text font of an item.
char *aw_get_window_font (qid)	Return the name of the text font of a window.
int aw_get_item_lowercase (qid)	Return the status of the text case mode of an item.
int aw_get_item_uppercase (qid)	Return the status of the text case mode of the item.
int aw_get_item_mixedcase (qid)	Return the status of the text case mode of an item.
int aw_get_item_integer (qid)	Return the status of integer validation of a text item.

Procedure Call	Description
int aw_get_item_real (qid)	Return the status of real number validation of a text item.
int aw_get_item_readonly (qid)	Return the read or write status of the item.
int aw_is_item_readonly (qid)	Check whether an item is in a read only state.
int aw_get_text_lowercase (qid)	Return the status of the text case mode of a text item.
int aw_get_text_uppercase (qid)	Return the status of the text case mode of a text item.
int aw_get_text_mixedcase (qid)	Return the status of the text case mode of a text item.
int aw_get_text_integer (qid)	Return the status of integer validation of a text item.
int aw_get_text_real (qid)	Return the status of real number validation of a text item.
int aw_get_text_readonly (qid)	Return the read or write status of a text item.
int aw_get_item_multiselect (qid)	Return the multiple selection status of an item.
int aw_is_item_multiselect (qid)	Return the single or multi mode of the item.
int aw_get_item_action (qid)	Return the action associated with an item.
int aw_get_item_visible (qid)	Return the visibility setting associated with an item.
int aw_is_item_visible (qid)	Check whether an item is visible.
int aw_get_item_active (qid)	Return the active status of an item.
int aw_is_item_active (qid)	Check whether an item is in an active state.
char *aw_get_active_id ( )	Return the name of the item that initiated the current action.
char *aw_get_item_caret ( )	Return the name of the item containing the text input caret.
char *aw_get_item_value (qid)	Get the current value of an item (number of rows or options for list or choice items in character format).
int aw_get_item_bit (qid, bit)	Get an option setting or a row number.
int aw_is_item_bit_set (qid, number)	Check if a row (`QLIST` item) or option (`QCHOICE` item) is selected.
char *aw_get_item_prompt (qid)	Return the text string for the prompt defined for an item.
char *aw_get_item_helpfile (qid)	Return the name of the helpfile referenced by an item.
int aw_get_window_stripe (qid)	Return the window stripe setting of an item a window.
char *aw_get_window_resource (qid)	Return the name of the resource file of a window.
int aw_get_state (char *qid)	Return the value of a control state.
char *aw_get_udata (char *qid)	Return user data.
int aw_get_menuitem_state (char *id)	Gets the state of a menu item.

Setting Item Values and Attributes

Procedure Call	Description
int aw_set_item_position (qid, x, y)	Reset the x and y location of an item.
int aw_set_item_xposition (qid, xvalue)	Reset the X location of an item.
int aw_set_item_yposition (qid, yvalue)	Reset the Y location of an item.
int aw_set_window_position (qid, x, y)	Set the position of a window.
int aw_set_item_width (qid, width)	Set the width of an item (not a window).
int aw_set_item_height (qid, height)	Set the height of an item (not a window).
int aw_set_item_cols (qid, cols)	Reset the width of an item.
int aw_set_item_rows (qid, rows)	Reset the height of an item.
int aw_set_window_width (qid, width)	Set the width of a window in pixels.
int aw_set_window_height (qid, height)	Set the height of a window in pixels.
int aw_set_item_depth (qid, depth)	Reset the depth of an item.
int aw_set_item_invert (qid, state)	Set or reset the invert state of an item.
int aw_set_item_etch (qid, value)	Set the etch value for an item to a specified number of pixels.
int aw_set_item_itemcolor (qid, color)	Set the color of an item.
int aw_set_item_backcolor (qid, color)	Set the background color of an item.
int aw_set_item_textcolor (qid, color)	Set the text color of an item.
int aw_set_item_selectcolor (qid, color)	Set the select color of an item.
int aw_set_item_greycolor (qid, color)	Set the item greyout color of an item.
int aw_set_item_label (qid, new_label)	Set the label of an item.
int aw_set_item_icon (qid, icon_name)	Set the icon associated with an item. The item must be visible and have a STATE,ICONIC attribute for the display to be updated.
int aw_set_item_dicon (qid, icon_name)	Specify or respecify the selected icon for an item.
int aw_set_item_font (qid, fontname)	Specify or respecify a font associated with an item.
int aw_set_item_lowercase (qid)	Set the text case mode of an item to convert input to lower case.
int aw_set_item_uppercase (qid)	Set the text case mode of an item to convert input to upper case.
int aw_set_item_mixedcase (qid)	Set the text case mode of the item to permit mixed case text.
int aw_set_item_integer (qid)	Set or reset the mode to validate input for an integer number.
int aw_set_item_real (qid)	Set the mode to validate input for a real number.
int aw_set_item_readonly (qid, state)	Set an item mode to allow or restrict modification of the contents.
int aw_set_text_lowercase (qid)	Set the mode to convert input to lower case characters.
int aw_set_text_mixedcase (qid)	Set the mode to convert input to mixed case characters.
int aw_set_text_integer (qid)	Set the mode to validate input for an integer number.
int aw_set_text_real (qid)	Set the mode to validate input for a real number.
int aw_set_text_readonly (qid, state)	Set or reset the read or write state of a text field.
int aw_set_item_multiselect (qid, on/off)	Set a list or choice item to single value or multi value selection.
int aw_set_item_action (qid, action_no)	Set the action number of an item.

Procedure Call	Description
int aw_set_item_visible (qid, state)	Reset the visibility state of an item.
int aw_set_item_active (qid, state)	Set or reset the active state of an item.
int aw_set_item_caret (qid)	Place the text input caret in a item.
int aw_set_item_value (qid, new_value)	Set the value of an item.
int aw_set_item_bit (qid, value)	Select a row (QLIST item) or option (QCHOICE item).
int aw_reset_item_bit (qid, value)	De-select a row (QLIST item) or option (QCHOICE item).
int aw_set_item_helpfile (qid, filename)	Assign or reassign the associated helpfile for an item.
int aw_set_item_prompt (qid, string)	Set or reset the prompt text string for an item.
int aw_prompt (prompt_string)	Display a prompt in the prompt field of the current window.
int aw_win_prompt (qid, prompt_string)	Display a prompt in the nominated prompt field of the window.
int aw_clear_prompt ( )	Remove any prompt text from the currently active prompt field.
int aw_message (message_string)	Display a message in the message field of the current window.
int aw_win_message (qid, message_string)	Display a message in the nominated message field of a window.
int aw_clear_message ( )	Remove any message from the currently active message field.
int aw_set_state (qid, state_value)	Sets the state of a menu item.

Updating Item Appearance

Procedure Call	Description
int aw_activate_item (qid)	Activate an item (cancel greyed-out state).
int aw_deactivate_item (qid)	Deactivate an item (grey-out the item).
int aw_show_item (qid)	Show an item (an entire window or specified items in a window).
int aw_show_item_relative (qid, relative_id, xoffset, yoffset)	Show an item relative to another item.
int aw_hide_item (qid)	Hide an item.
int aw_hide_all_popups ( )	Hide all popup windows from the screen.
int aw_open_window (qid)	Open a window from an icon.
int aw_close_window (qid)	Close a window to an icon.
int aw_refresh_window (qid)	Force AccessWare to read the resource file for a window and refresh its content.
int aw_set_window_resource (qid, resource_file)	Redefine a window from the contents of a new resource file.

Procedures Associated with Lists

Procedure Call	Description
int aw_get_list_width (qid)	Get the maximum number of columns in a list contents file.
int aw_get_list_length (qid)	Get the number of rows in a list.
int aw_mark_list_line (qid, line_no, char_pos, mark)	Write character mark at character position char_pos.
char *aw_get_list_line (qid)	Get the text of the currently selected row of a list item.
char *aw_get_line (qid, lineno)	Get the row text at lineno of a list item.
int aw_get_scroll_char (qid)	Get the number of columns by which the list has been scrolled.
int aw_get_scroll_line (qid)	Get the number of rows by which the list has been scrolled.
int aw_scroll_to_line (qid, line_no)	Vertically scroll a list item to the given line number.
int aw_scroll_to_char (qid, char_no)	Horizontally scroll a list item to the character column number.
int aw_add_line_to_list (qid, line, pos)	Add a new line to a list item.
int aw_delete_line_from_list (qid, line, pos)	Delete an existing line from a list item at a given row position.
char *aw_get_list_file (qid)	Get the name of the file associated with a list.
char *aw_get_list_heading (qid)	Return the heading associated with a list item.
int aw_set_item_heading (qid, heading)	Set or reset the text for the header item associated with a list item.
int aw_set_list_heading (qid,string)	Set or reset the heading of a list item.
int aw_refresh_list (qid)	Force an update of a displayed list from contents held in memory.
int aw_update_list (qid)	Force AccessWare to read the listfile associated with a list item again.
int aw_save_listfile (qid, listfile)	Save the contents of a list in a file.

Procedure Call	Description
int aw_reset_list (qid)	Clear the list contents on the screen and delete the disk file associated with the list.
int aw_set_list_file (qid, listfile)	Reset the pathname of the list file to be used in an item (this could be a list contents or a menu contents).
int aw_switch_list_font (qid, fontname)	Reset the font in which a list is displayed.
int * aw_get_list_selection (qid)	Return the row numbers of the rows that are selected from the list.

Procedures Associated with Tree Nodes

Procedure Call	Description
int aw_get_node_selected (qid, line_no_of_listfile)	Check if a tree node is selected.
int aw_get_node_highlighted (qid, line_no_of_listfile)	Get the highlight color of a tree node.
int aw_get_node_visible (qid, line_no_of_listfile)	Check if a tree node is visible.
int aw_is_node_selected (qid, line_no_of_listfile)	Check if a tree node is selected.
int aw_is_node_highlighted (qid, line_no_of_listfile)	Check if a tree node is highlighted.
int aw_is_node_visible (qid, line_no_of_listfile)	Check if a tree node is visible.
int aw_get_node_textcolor (qid, line_number)	Return the color of the text of a tree node.
int aw_get_node_linkcolor (qid, line_no_of_listfile)	Get the link color of a tree node.
int aw_get_node_access (qid, line_no_of_listfile)	Get the color of the access indicator of a tree node.
int aw_set_node_selected (qid, line_no_of_listfile, sel_state)	Set a tree node to the selected state.
int aw_set_node_highlight (qid, line_no_of_listfile, color)	Highlight a tree node in a specified color.
int aw_set_node_visible (qid, line_no_of_listfile, state, repaint)	Set a tree node to visible or hidden.
int aw_set_node_linkcolor (qid, line_no_of_listfile, color)	Set the color of the link between a tree node and its parent.
int aw_set_node_access (qid, line_no_of_listfile, color)	Set the access indicator color of a tree node.
aw_create_cgm_file (tree_qid, cgm_filename, scale, paper_size, all_tiers, tier_from, tier_to, is_multiple_cgm_pages)	Creates a binary CGM format file of a tree.

Miscellaneous Procedures

Procedure Call	Description
int aw_add_item_to_group (group,qid)	Add an item to an existing group.
int aw_del_item_from_group (group,qid)	Delete an item from a group.
int aw_get_cursor_position (x, y)	Return the position of the mouse cursor from the top left corner of the AccessWare window in which the last event occurred.
int aw_cursor_into_item (qid)	Move the mouse cursor to an item.
int aw_start_macro ( )	Start a macro process and mouse cursor changes to busy.
int aw_exit_macro ( )	Return the application to normal operating mode. Mouse cursor returns to normal mode.

Procedure Call	Description
int aw_get_global (global, string)	Return the expansion of the named global variable.
int aw_set_global (global, string)	Sets the value of the defined global variable
char *aw_search_path (file)	Search through the search path to find the first occurrence of a file.
int aw_perl_exec ( )	Execute a specified `perl` script.
int aw_system (call)	Pass the string to the native operating system for processing.
int aw_to_cadds (cmd)	Pass the string to CADDS for processing.
void aw_beep ( )	Sound the bell on the PC or workstation being used.
int aw_register_repository (char* filename)	Registers a file as a repository other than the GLOBVARS files
int aw_register_repository_latest (file)	Registers the latest message file.

AccessWare Programming Examples

This section contains a number of examples on writing AccessWare actions in C code. The same examples as in Chapter 5, "AccessWare Script Interface." are used to illustrate the techniques. The difference is that the ACTIONs are implemented by C programs instead of operating system scripts. Hence, the ACTION attributes in the resource files take a numeric integer as their argument instead of a Script action.

Please note: In each of the following examples, the source code is contained in two files:

- main.c
- aw_user_exit.c

The files are compiled and linked to form an executable named example. The makefile to create a Solaris executable for these examples is as follows:

```
#
# Makefile for example programs
#
LIBS = -lX11 -lcvawtools -lcvawx11
CFLAGS = -O -I$(EPD_HOME)/include -I$(EPD_HOME)/include/aw ->
-I/usr/openwin/include -L$(EPD_HOME)/lib -L/usr/openwin/lib ->
        -DX11 -DMOTIF -DNO_CVKERNEL
OBJS = main.o aw_user_exit.o
#
example : $(OBJS)
          cc $(OBJS) $(CFLAGS) $(LIBS) -o example
main.o :  main.c
          cc $(CFLAGS) -c main.c -DMAIN
aw_user_exit.o : aw_user_exit.c
                 cc $(CFLAGS) -c aw_user_exit.c
```

The preceding makefile can be used with any of the examples in the following sections.

Sample makefiles are provided on your release tape under the directory $EPD_HOME/obj.

Passing Arguments to a Program

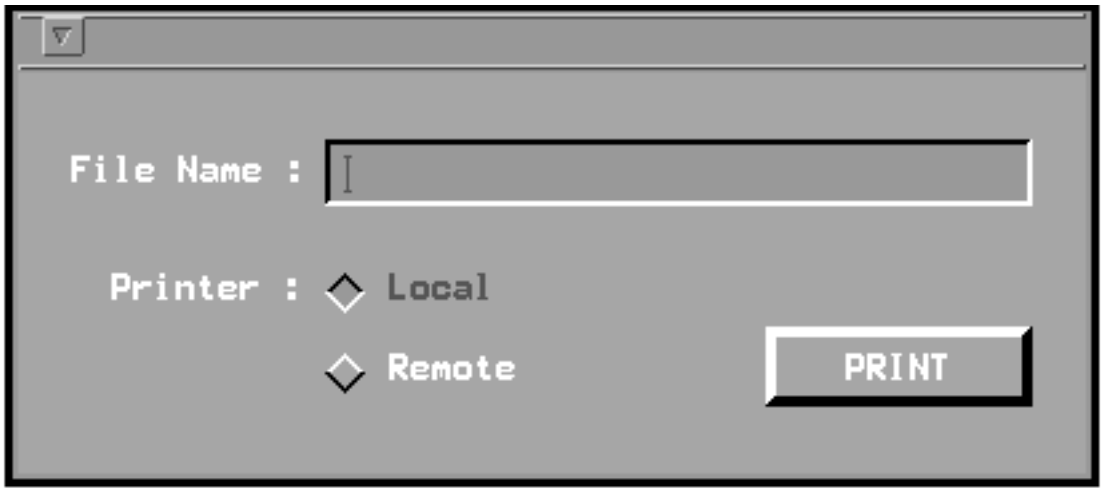
This example sends a specified file to a local or remote printer. The resource file for this example is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,150,LABEL,Print Window,QEND
```

```
# File Name input text field
QMESSAGE
XPOS,20,YPOS,31,COLS,10,ROWS,1,LABEL,File Name:,DEPTH,0,QEND
QTEXT
QID,filename,XPOS,115,YOFFSET,-6,WIDTH,265,HEIGHT,25,INVERT,1,QEND
# Printer fields
QMESSAGE
XPOS,35,YPOS,75,COLS,9,ROWS,1,LABEL,Printer:,DEPTH,0,QEND
QCHOICE
QID,printer,XPOS,115,YOFFSET,0,WIDTH,150,HEIGHT,30, ->
STYLE,TOGGLE,LAYOUT,V,LABEL,Local~Remote~QQ,VALUE,1,QEND
# The PRINT Button
QBUTTON
XPOS,280,YPOS,95,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,PRINT, ->
ACTION,1000,QEND
# ============================================================
QATTREND
```

This resource file contains a single `ACTION` defined against the PRINT button. This is identical to the resource file in Chapter 5, “AccessWare Script Interface.” apart from the `ACTION` definition on the PRINT button. The panel for this example is shown in the following figure.

Figure 6-3 Example of a Simple File Print Panel



The source code for the `main.c` procedure is as follows:

```
/*******************************
**  AccessWare main procedure **
*******************************/

#include ”access.h”

main(argc, argv)

int argc;char **argv;
```

```
{
    char command_file[255];
    if ( getenv("AW_RESOURCE") == NULL )
    {
       if ( argc != 2 )
       {
          fprintf(stderr,"usage: %s <res file>\n",argv[0]);
          exit(1);
       }
       strcpy(command_file,argv[1]);
                                  /* Command line input */
    }
    else
    {
       strcpy(command_file,getenv("AW_RESOURCE"));
                                /* Environment variable */
    }

    /* Pass control to the main AccessWare program */
    aw_init_windows(command_file);

    exit();
}
```

This main program requires the name of the resource file to be entered as either an argument to the resulting executable or within an environment variable named `AW_RESOURCE`. It then calls the procedure `aw_init_windows` to pass control to the AccessWare program.

The `aw_user_exit.c` source file contains:

```
/************************************************
** File: aw_user_exit()                          **
** Purpose : Example for AccessWare application **
************************************************/

#include <stdio.h>
#include "access.h"
extern char *aw_get_item_value();

aw_user_exit( reg, user_action, value, mouse )

int reg, user_action, value, mouse;
{
    int pick;            /* Defines which printer to use */
    char pname[10];      /* Holds name of printer to use */
    char filename[255];  /* Filename in 'filename' field */
    char command[255]; /* String to hold 'print command' */

    switch (user_action)
    {
    /* Action 1000 called from PRINT button */
```

```
    case 1000:
        if (mouse == MOUSE_UP)
        {
           /* Get the value of the QCHOICE printer item
              A value of 1 is the remote printer(lw1),
               2 is the local printer(lw2) */
           pick = atoi(aw_get_item_value("printer"));

           if (pick == 1)
               strcpy(pname,"lw1");
           else
               strcpy(pname,"lw2");

           /* Get filename entered in the filename field */
           strcpy(file_name,aw_get_item_value("filename"));

           /* Build up print command to issue to system  */
           sprintf(command,"lpr -P%s %s\0",pname,filename);
           system(command);
        }
    break;
    }
}
```

Please note: The use of the AccessWare procedure calls to `aw_get_item_value`.

Updating a Text Field

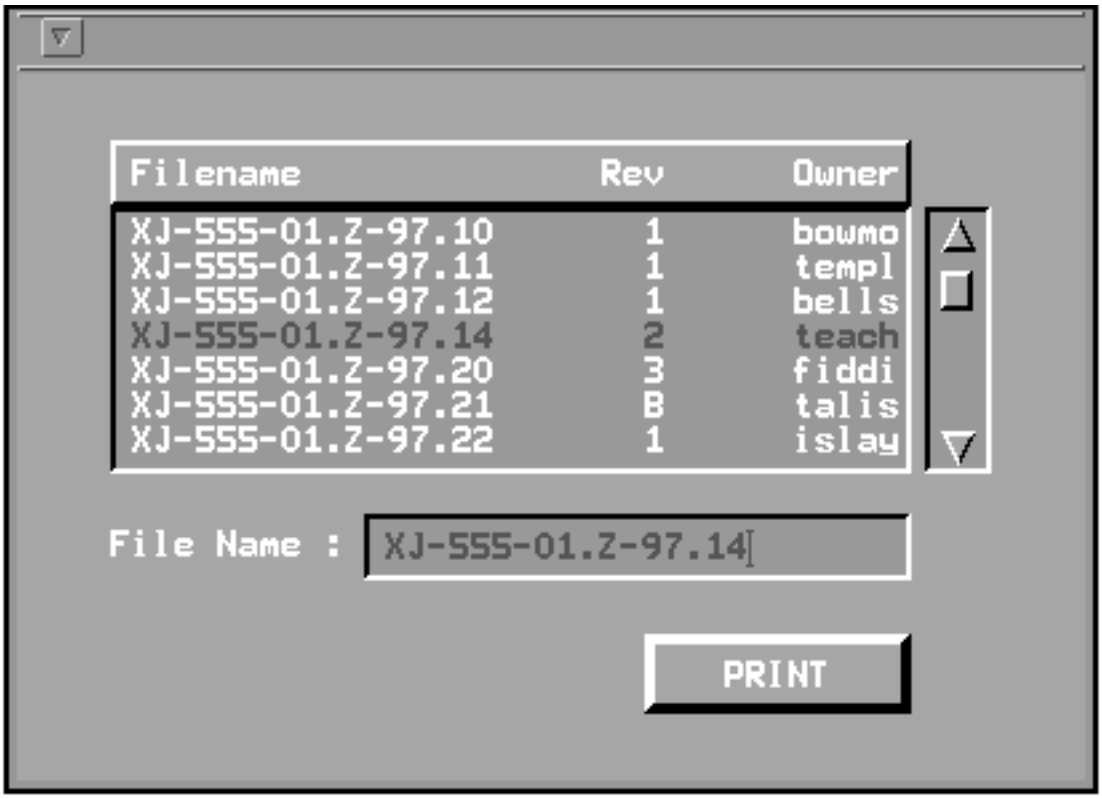
This example enables the user to select an item from a list. It is then used to update the contents of an input text field. The resource file for this interface with actions to execute the required pieces of code is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,265,LABEL,File List,QEND
# File List Field
QLIST
QID,filelist,XPOS,35,YPOS,50,WIDTH,300,HEIGHT,100,INVERT,1, ->
LISTFILE,mylist,ACTION,1010,QEND
# Vertical Scrollbar for List
QPANEL
XAFTER,5,YOFFSET,0,HEIGHT,100,WIDTH,25,INVERT,1, ->
ACTION,VS,filelist,VALUE,2,QEND
# List Header Bar
QPANEL
XPOS,35,YPOS,25,HEIGHT,25,WIDTH,300,ACTION,HEAD,filelist,QEND
# File Name input text field
QMESSAGE
XPOS,35,YPOS,171,COLS,10,ROWS,1,DEPTH,0,LABEL,File Name :,QEND
QTEXT
QID,filename,XPOS,130,YOFFSET,-6,WIDTH,205,HEIGHT,25,INVERT,1,QEND
# PRINT Button
```

```
QBUTTON
XPOS,235,YPOS,210,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,PRINT, ->
ACTION,1020,QEND
# ============================================================
QATTREND
```

The following figure shows the panel created by the resource file.

Figure 6-4 Example of a List and Print Panel



The same `main.c` procedure as defined in the previous example can be used. However, the `aw_user_exit.c` code needs to define actions `1010` and `1020` from the resource file. The `aw_user_exit.c` file now contains the following:

```
/***********************************************
** File: aw_user_exit()                       **
** Purpose : Example for AccessWare application **
***********************************************/

#include <stdio.h>
#include "access.h"
extern char *aw_get_item_value(), *aw_get_list_line();
extern int aw_set_item_value();

aw_user_exit( reg, user_action, value, mouse )
```

```
int reg, user_action, value, mouse;
{
    char filename[255]; /* Filename entered in 'filename'
                                field or picked from list */
    char command[255]; /* String to hold 'print command' */

    switch (user_action)
    {
    /* Action 1010 called from LIST item */
    case 1010:
        if (mouse == MOUSE_UP)
        {
           /* Get 'picked' filename from list */
           strcpy(filename,aw_get_list_line("filelist"));
           /* Put filename into "filename" input field */
           aw_set_item_value("filename", filename);
        }
        break;

    /* Action 1020 called from PRINT button */
    case 1020:
        if (mouse == MOUSE_UP)
        {
           /* Get filename entered in the filename field */
           strcpy(filename,aw_get_item_value("filename"));

           /* Build up print command to issue to system */
           sprintf(command,"lpr -Plw2 %s\0",filename);
           system(command);
        }
        break;
    }
}
```

Please note: The use of the AccessWare procedure calls to `aw_get_list_line` and `aw_set_item_value`.

Updating List Contents

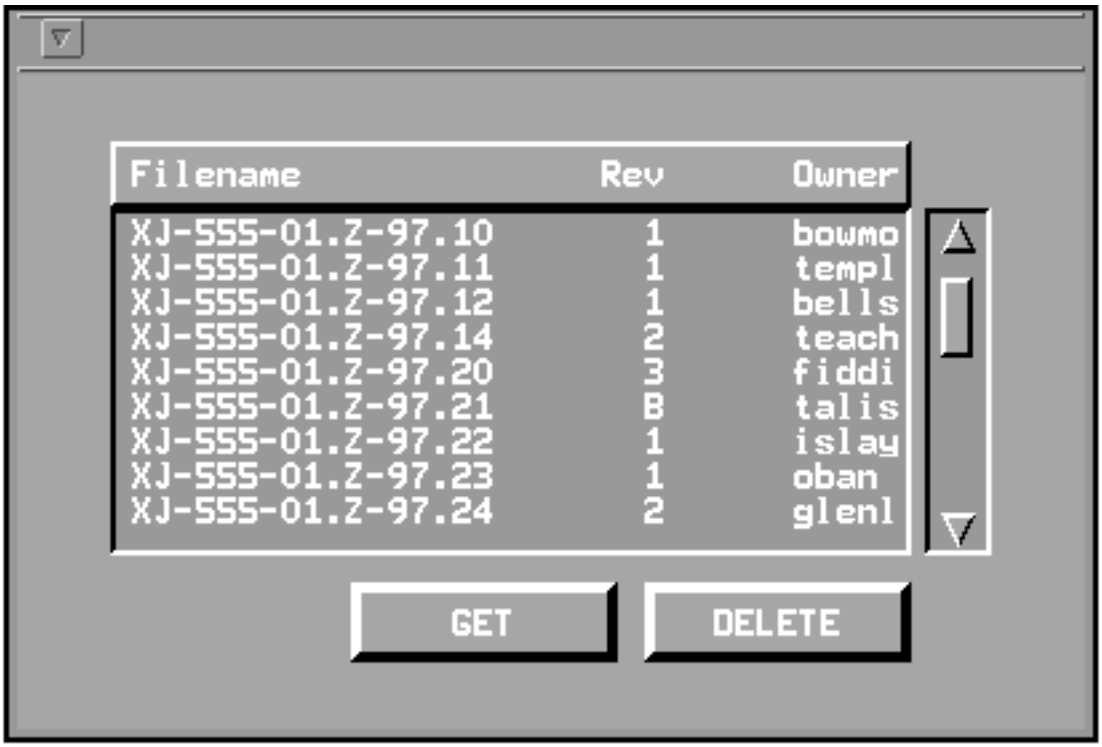
This example displays a panel that enables a user to select an item from a list, then perform an action on it. After the action has been completed the list must be updated. The resource file for this example is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,245,LABEL,File List,QEND
# File List Field
QLIST
QID,filelist,XPOS,35,YPOS,50,WIDTH,300,HEIGHT,130,INVERT,1, ->
LISTFILE,mylist,ACTION,CF,myactions,QEND
# Vertical Scrollbar for List
QPANEL
```

```
XAFTER,5,YOFFSET,0,HEIGHT,130,WIDTH,25,INVERT,1, ->
ACTION,VS,filelist,VALUE,2,QEND
# List Header Bar
QPANEL
XPOS,35,YPOS,25,HEIGHT,25,WIDTH,300,ACTION,HEAD,filelist,QEND
# GET Button
QBUTTON
XPOS,125,YPOS,190,WIDTH,100,HEIGHT,30,DEPTH,4, ->
LABEL,GET,ACTION,1030,QEND
# DELETE Button
QBUTTON
XPOS,235,YPOS,190,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,DELETE, ->
ACTION,1040,QEND
# ============================================================
QATTREND
```

The panel created by the preceding resource file is shown in the following figure.

Figure 6-5 Example of a List Requiring an Update Action



The item of special interest in this example is the `QBUTTON` labelled . The `QLIST` must be updated once the delete actions are completed.

In this example, the main procedure has been altered to perform two operations:

- Create an up-to-date list file (`mylist`) to be displayed in the interface.
- Omit the need to specify the resource file name. The resource file name is `example3.res`

The source code for the `main.c` procedure is the following:

```
/*****************************
**  AccessWare main procedure **
*****************************/

#include "access.h"

main(argc, argv)

int argc;
char **argv;
{
    /* Delete the current list file */
    unlink("mylist");

    /* Regenerate the list file */
    system("ls -C1 >mylist");

    /* Pass control to the main AccessWare program */
    aw_init_windows(command_file);

    exit();
}
```

The `aw_user_exit.c` file for this example shows how to force an update of the list contents to the screen:

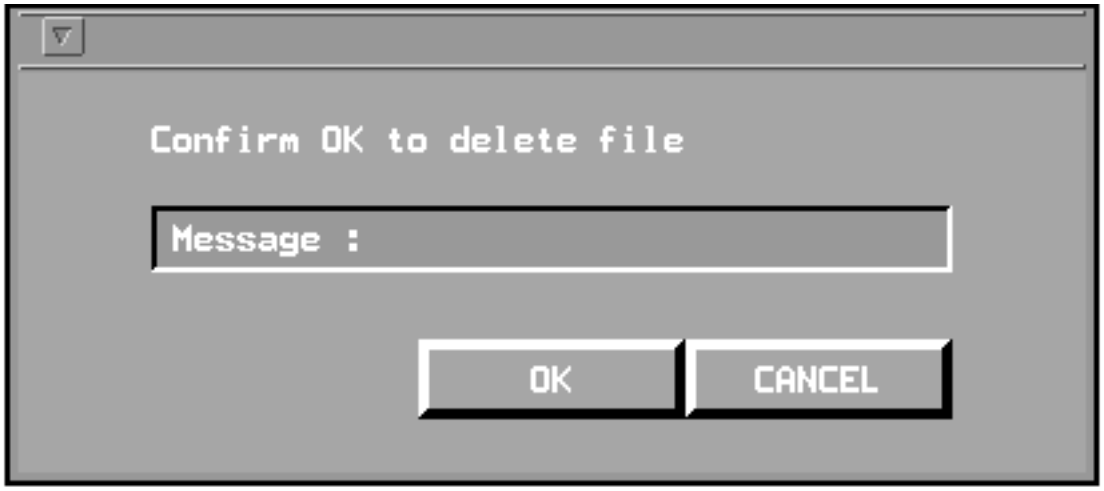
```
/***********************************************
** File: aw_user_exit()                       **
** Purpose : Example for AccessWare application **
***********************************************/

#include <stdio.h>
#include "access.h"
extern char *aw_get_item_value(), *aw_get_list_line();
extern int aw_set_item_value(), aw_update_list();

aw_user_exit( reg, user_action, value, mouse )

int reg, user_action, value, mouse;
{
    char filename[255];   /* File name picked from list */
    char command[255]; /* String to hold 'print command' */

    switch (user_action)
```

```
    {
    /* Action 1030 called from GET button */
    case 1030:
        if (mouse == MOUSE_UP)
        {
            /* Get 'picked' filename from list */
            strcpy(filename,aw_get_list_line("filelist"));
            /* Do GET action - whatever that is! */
        }
        break;

    /* Action 1040 called from DELETE button */
    case 1040:
        if (mouse == MOUSE_UP)
        {
           /* Get filename entered in the filename field */
           strcpy(filename,aw_get_list_line("filelist"));
           /* Remove newline from end of returned string */
           if (filename[strlen(filename)-1] == '\n')
                filename[strlen(filename)-1] = '\0';

           /* Delete the specified file */
           unlink(filename);
           /* Delete the current list file */
           unlink("mylist");
           /* Regenerate the list file */
           system("ls -C1 >mylist");

           /* Force update of displayed list contents
              this will now reflect deleted file */
           aw_update_list("filelist");
        }
        break;
    }
}
```

Creating a Confirmation Panel

This example describes how to define and display a confirmation window prior to deleting the file.

Figure 6-6 Example of Confirmation Panel



The definition of the confirmation window is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,150,LABEL,Confirmation,QEND
# Output confirm message
QMESSAGE
XPOS,50,YPOS,20,WIDTH,300,HEIGHT,25,LABEL,Confirm OK to delete
file,QEND
QPANEL
QID,conf_msg,XPOS,50,YPOS,50,WIDTH,300,HEIGHT,25,INVERT,1, ->
LABEL,Message : ,QEND
# The CANCEL Button
QBUTTON
XPOS,250,YPOS,100,WIDTH,100,HEIGHT,30,DEPTH,4  ->
LABEL,CANCEL,ACTION,1050,QEND
# The CONFIRM Button
QBUTTON
XPOS,150,YPOS,100,WIDTH,100,HEIGHT,30,DEPTH,4,LABEL,OK,->
ACTION,1060,QEND
# ================================================
QATTREND
```

There is no need to update the `main.c` program from the previous example. However, the `aw_user_exit.c` code must display the confirmation window prior to deleting the file or quitting from the request.

The `aw_user_exit.c` file for this example is as follows:

```
/***********************************************
** File: aw_user_exit()                        **
** Purpose : Example for AccessWare application **
***********************************************/
```

```
#include <stdio.h>
#include "access.h"
extern char *aw_get_item_value(), *aw_get_list_line();
extern int aw_set_item_value(), aw_update_list();

aw_user_exit( reg, user_action, value, mouse )

int reg, user_action, value, mouse;
{
    static char filename[255];     /* Filename from list */
    char command[255]; /* String to hold 'print command' */

    switch (user_action)
    {
    /* Action 1030 called from GET button */
    case 1030:
        if (mouse == MOUSE_UP)
        {
           /* Get 'picked' filename from list */
           strcpy(filename,aw_get_list_line("filelist"));

           /* Do GET action - whatever that is! */
        }
        break;

    /* Action 1040 called from DELETE button */
    case 1040:
        if (mouse == MOUSE_UP)
        {
           /* Get filename entered in the filename field */
           strcpy(filename,aw_get_list_line("filelist"));
           /* Remove newline from end of returned string */
           if (filename[strlen(filename)-1] == '\n')
              filename[strlen(filename)-1] = '\0';
           /* Put file name into confirm window */
           aw_set_item_value("conf_msg",filename);
           /* Display the confirm window */
           aw_show_item("confirm");
        }
        break;

    /* Action 1050 called from CANCEL button */
    case 1050:
        if (mouse == MOUSE_UP)
        {
           /* Remove the confirm window */
           aw_hide_item("confirm");
        }
        break;

    /* Action 1060 called from OK button */
    case 1060:
        if (mouse == MOUSE_UP)
```

```
        {
           /* Delete the specified file */
           unlink(filename);
           /* Delete the current list file */
           unlink("mylist");
           /* Regenerate the list file */
           system("ls -C1 >mylist");

           /* Remove the confirm window */
           aw_hide_item("confirm");

           /* Force update of displayed list contents
              this will now reflect deleted file */
           aw_update_list("filelist");
        }
        break;
    }
}
```

Please note: The control must return from the `aw_user_exit` procedure in order to obtain further input. Therefore, the filename defined must be stored in static memory to ensure that it is still available on the next entry into the `aw_user_exit` procedure.

Scrolling to the Matching Line

You can enable scrolling to the matching line in the list based on input in the textbox or the list. In the resource file of a window, a `QTEXT` and a `QBUTTON` are described as follows:

```
QTEXT
QID,mytext,XOFFSET,110,YOFFSET,0,WIDTH,290,MAXCOLS,90,HEIGHT,
25,INVERT,1,DEPTH,1,STRETCHX,100,MOVEY,100,MOVEX,100,QEND
QBUTTON
QID,CA_DOMAIN_BUT,XAFTER,4,YOFFSET,2,WIDTH,22,HEIGHT,2,VISIBLE,1,
LABEL,...,ACTION,7009,STRETCHX,100,MOVEY,100,MOVEX,100,QEND
```

The resource file for a `QLIST` is as follows:

```
QLIST
QID,project_browser_list,FONT,$CA_LIST_FONT,XPOS,0,YPOS,0,WIDTH,
260,HEIGHT,102,LISTFILE,$CA_PROJECTLIST,DEPTH,1,ACTION,8639,INVER
T,1,STRETCHX,100,STRETCHY,100,MOVEX,100,MOVEY,100,QEND
```

When you click the button adjacent to the textbox, the list is populated and the `QTEXT qid` is stored in the `browser_list` variable.

```
Action 7009

/*Write the list file*/
```

```
aw_update_list("project_browser_list");

cvstropy(browser_list, "mytext");

aw_show_item_relative("CA_PROJECT_BROWSER", from_where, 0, 25);

aw-cursor_into_item("project_browser_list");
```

On selection in the List, the Textbox is populated as follows:

```
Action 8639

ptr = aw_get_list_line("project_browser_list");

if (!ptr)

      return (FALSE);

if (*ptr == '\0')

      return (FALSE);

if (!cvstrcmp(browser_list, "my_text"))

aw_set_item_value("my_text",ptr)
```

Chapter 7

User Interface Examples

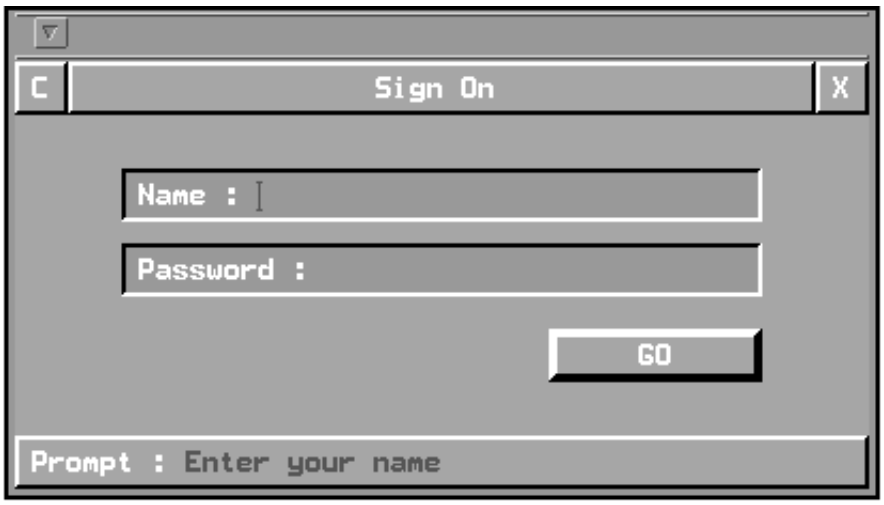
This chapter provides some examples of AccessWare interfaces. It takes you through the construction of the windows and shows how various elements are defined.

- Example of Text Fields and Buttons
- Example of Lists and Choice Items
- Example of a Menu Driven Interface
- Example of Extended Menus

Example of Text Fields and Buttons

This example defines the simple window to accept a user name and password, shown in Chapter 1, “Introduction to AccessWare”.

Figure 7-1 Simple dialog box accepting input



This dialog box is used to log the user on to the application and then open the main application windows.

The Window

The window definition need only define the window size and its position on the screen. Use the QWINDOW item and locate the window 100 pixels from the left side of the screen and 200 pixels from the top of the screen. A useful window size is a width of 400 pixels and height of 200 pixels.

Example:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,200,LABEL,Sign On,QEND
```

The Header

A window often requires some form of header that contains:

- A title
- A button to close the window to an icon
- An exit button.

These items are no different from any other items that you define to appear within a window. Use a common style of header for all your windows.

A sample header follows. In this example, the header contains a title bar and buttons to close the window to an icon (labelled C) and to exit from the application (labelled X).

Figure 7-2 Example of a header

The header definition for this example is:

```
QBUTTON
XPOS,0,YPOS,0,WIDTH,25,HEIGHT,25, ->
LABEL,C,PROMPT,Turn the window into an Icon,ACTION,CL,QEND
QPANEL
XPOS,25,YPOS,0,WIDTH,350,HEIGHT,25,LABEL, Sign On,QEND
QBUTTON
XPOS,375,YPOS,0,WIDTH,25,HEIGHT,25,   ->
LABEL,X,PROMPT,Exit from this application,ACTION,EX,QEND
```

Please note: The AccessWare `ACTIONs` (`CL` and `EX`) are used for the icon and exit buttons.

The Footer

A prompt area is placed at the bottom of a window to provide a consistent place for user Help information.

A sample footer window is shown in the following figure:

Figure 7-3 Example of a footer window

The definition of the prompt area is:

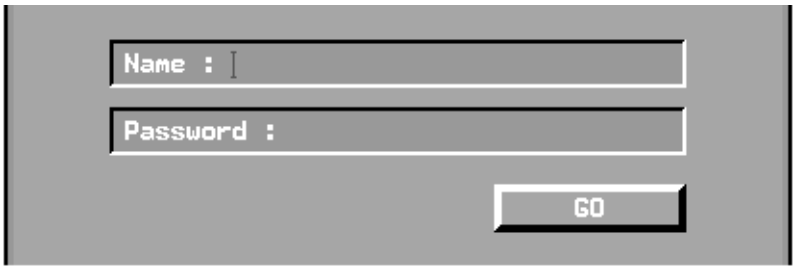
```
QPANEL
XPOS,0,YPOS,175,WIDTH,400,HEIGHT,25,LABEL,Prompt:, ACTION,PR,QEND
```

Please note: The AccessWare action of `PR` defines the special use of the `QPANEL` item, that is, it is used as a prompt field.

Window Fields and Buttons

The following window example contains two input fields and one button.

Figure 7-4 Example of a window



The input fields are defined as QTEXT items. The field for the user name is defined as follows:

```
QTEXT
QID,sgn_name,XPOS,50,YPOS,50,WIDTH,300,HEIGHT,25,INVERT,1, ->
LABEL,Name: ,PROMPT,Enter your name,QEND
```

The INVERT attribute makes the field appear recessed into the screen, which indicates that it is an input field. (The DEPTH attribute default sets the recessed depth at 2 pixels.)

The password field is defined in a similar way. However, the input characters must be suppressed and replaced by asterisks (*). This is achieved by adding STATE, SECURE to the definition, which echoes the user's keystrokes as asterisks.

Example:

```
QTEXT
QID,sgn_password,XPOS,50,YPOS,85,WIDTH,300,HEIGHT,25,INVERT,1, ->
LABEL,Password: ,STATE,SECURE,PROMPT,Enter your password,QEND
```

The GO button is used to check the user's id and password so as to gain entry to the main application functions. This checking must be done by a piece of application code or script. In this example, the action is executed by C code identified by an ACTION labelled 1000.

Example:

```
QBUTTON
XPOS,250,YPOS,125,WIDTH,100,HEIGHT,25,DEPTH,4, ->
LABEL,GO,PROMPT,Enter the main application,ACTION,1000,QEND
```

The Complete Resource File

From the elements of the window defined so far, the complete resource file for this window follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,200,LABEL,Sign On,QEND
# ================================================
QBUTTON
XPOS,0,YPOS,0,WIDTH,25,HEIGHT,25, ->
LABEL,C,PROMPT,Turn the window into an Icon,ACTION,CL,QEND
QPANEL
XPOS,25,YPOS,0,WIDTH,350,HEIGHT,25,LABEL,Sign On,QEND
QBUTTON
XPOS,375,YPOS,0,WIDTH,25,HEIGHT,25,   ->
LABEL,X,PROMPT,Exit from this application,ACTION,EX,QEND
# ================================================
# User name input text field
QTEXT
QID,sgn_name,XPOS,50,YPOS,50,WIDTH,300,HEIGHT,25,INVERT,1,  ->
LABEL,Name : ,PROMPT,Enter your name,QEND
# User password input field
QTEXT
QID,sgn_password,XPOS,50,YPOS,85,WIDTH,300,HEIGHT,25,INVERT,1, ->
LABEL,Password : ,STATE,SECURE,PROMPT,Enter your password,QEND
# Action Button
QBUTTON
XPOS,250,YPOS,125,WIDTH,100,HEIGHT,25,DEPTH,4,  ->
LABEL,GO,PROMPT,Enter the main application,ACTION,1000,QEND
# ================================================
# Define the footer prompt area
QPANEL
XPOS,0,YPOS,175,WIDTH,400,HEIGHT,25,LABEL,Prompt : ,ACTION,PR,QEND
# End of window definition
QATTREND
# ================================================
```

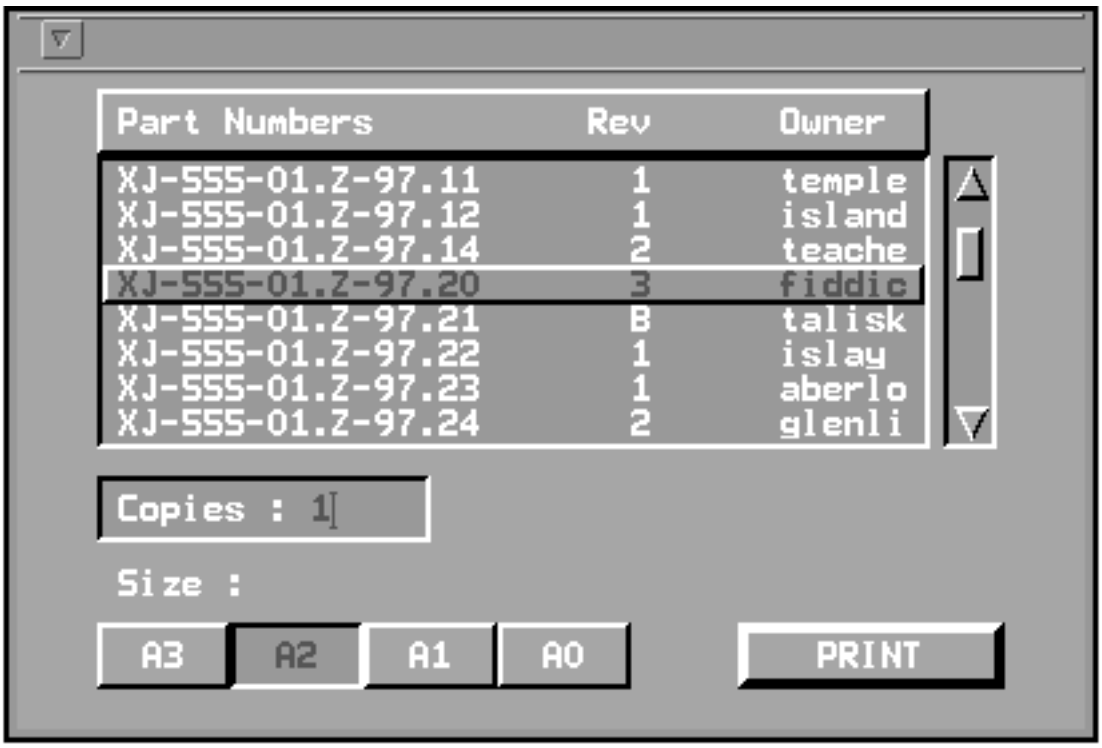
Example of Lists and Choice Items

This example describes how to create a window and introduces two other item types:

- `QLIST` - to display a list of filenames
- `QCHOICE` - to define a set of choice buttons

The example creates a window to select a file for printing as seen in the following figure.

Figure 7-5 Example of a window defined by `QLIST` and `QCHOICE` items



The way to reference this window from the previous Sign on example is explained later.

LIST Items

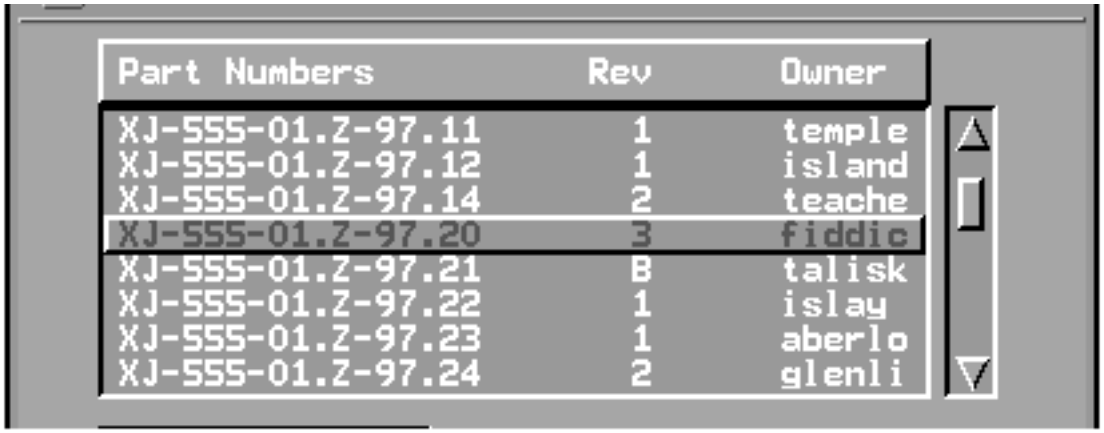
The list items shown above consist of a list panel with a header bar and a scrollbar. The definitions for these items are:

Example:

```
QLIST
QID,list1,XPOS,30,YPOS,30,COLS,37,ROWS,8,INVERT,1,LISTFILE,mylist
,QEND
QPANEL
XOFFSET,0,YOFFSET,-25,COLS,37,HEIGHT,25,ACTION,HEAD,list1,QEND
QPANEL
XAFTER,5,YOFFSET,25,WIDTH,20,HEIGHT,110,INVERT,1, ->
ACTION,VS,list1,VALUE,2,QEND
```

The items created by the preceding definitions are shown in the following figure.

Figure 7-6 Example of a window defined using `QLIST` and `QPANEL` items



Please note: The action definitions `VS` (vertical scroll bar) and `HEAD` (header bar) reference the `QLIST QID` name (list1) in both the `QPANEL` items.

AccessWare searches the `LISTFILE` directory for a header definition file associated with the `LISTFILE` name. In this example, it should find a file named `mylist.hdg` that contains a single row defining a Part Number, Revision and Owner.

The Input Field

The definition for the input field to enter the number of copies is as follows:

Example:

```
QTEXT
QID,copies,XPOS,30,YPOS,150,WIDTH,125,HEIGHT,25,INVERT,1, ->
LABEL,Copies : ,PROMPT,Enter number of copies to be
printed,VALUE,1,QEND
```

The QTEXT item created by this definition is shown in the following figure.

Figure 7-7 Example of a window defined using the QTEXT item

The field requesting the number of copies is just like those defined in the Sign on window, but with the addition of a default VALUE of 1.

QCHOICE Item

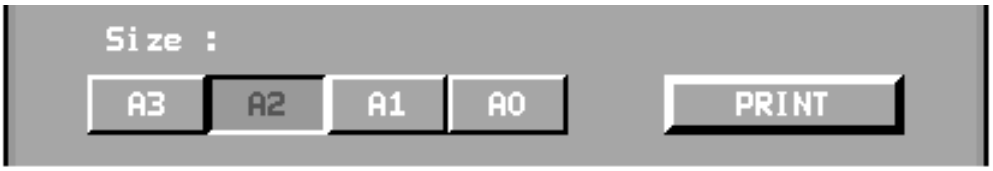
The QCHOICE item provides a choice of printing sizes at A3, A2, A1, or A0, with the default set to A2. A message item is used to output the Size text string to label the row of buttons. The depth of the QMESSAGE item is '0' since the text field is not an input field and should therefore appear neither recessed nor raised.

Example:

```
QMESSAGE
XPOS,38,YPOS,185,WIDTH,50,HEIGHT,25,LABEL,Size:,QEND
QCHOICE
QID,mychoice,XPOS,30,YPOS,205,WIDTH,50,HEIGHT,25, ->
LABEL,A3~A2~A1~A0~QQ,VALUE,2,QEND
```

The items created by these definitions, together with the PRINT button is shown in the following figure.

Figure 7-8 Example of a window defined using the QCHOICE **item**



Please note: The QMESSAGE item does not need to be referenced from the application or the interface and therefore does not have a QID.

The Complete Resource File

The complete resource file for the window can be built from the elements defined above. The definition of a QBUTTON to provide the PRINT button that actions the Print function is also included.

```
QWINDOW
XLOC,100,YLOC,100,HEIGHT,245,WIDTH,400,LABEL,Print Window,QEND
# ================================================
QLIST
QID,list1,XPOS,30,YPOS,30,COLS,37,ROWS,8,INVERT,1,LISTFILE, ->
mylist,QEND
QPANEL
XOFFSET,0,YOFFSET,-25,COLS,37,HEIGHT,25,ACTION,HEAD,list1,QEND
QPANEL
XAFTER,5,YOFFSET,25,WIDTH,20,HEIGHT,110,INVERT,1, ->
ACTION,VS,list1,VALUE,2,QEND
QTEXT
QID,copies,XPOS,30,YPOS,150,WIDTH,125,HEIGHT,25,INVERT,1, ->
LABEL,Copies:,PROMPT,Enter number of copies to be
printed,VALUE,1,QEND
QMESSAGE
XPOS,38,YPOS,185,WIDTH,50,HEIGHT,25,LABEL,Size:,QEND
QCHOICE
QID,mychoice,XPOS,30,YPOS,205,WIDTH,50,HEIGHT,25, ->
LABEL,A3~A2~A1~A0~QQ,VALUE,2,QEND
QBUTTON
QID,print,XPOS,270,YPOS,205,WIDTH,100,HEIGHT,25,DEPTH,4, ->
LABEL,PRINT,ACTION,1020,QEND
# ================================================
QATTREND
```

References to Other Windows

The resource file for the first window displayed must contain a list of the other windows directly invoked from it and defined in other resource files.

For example, if the Sign on window is the first window, it is defined in the main resource file. If a second window named `print_window` is invoked from the Sign on window and defined in another resource file named `print_win.res`, the main resource file must contain an additional line as follows:

Example:

```
QPOPUP
QID,print_window,LABEL,print_win.res,VISIBLE,0,QEND
```

This defines `print_window` as the name of the second window and `LABEL` identifies the pathname of its resource file. The name `print_window` can now be used within action code for manipulating the window.

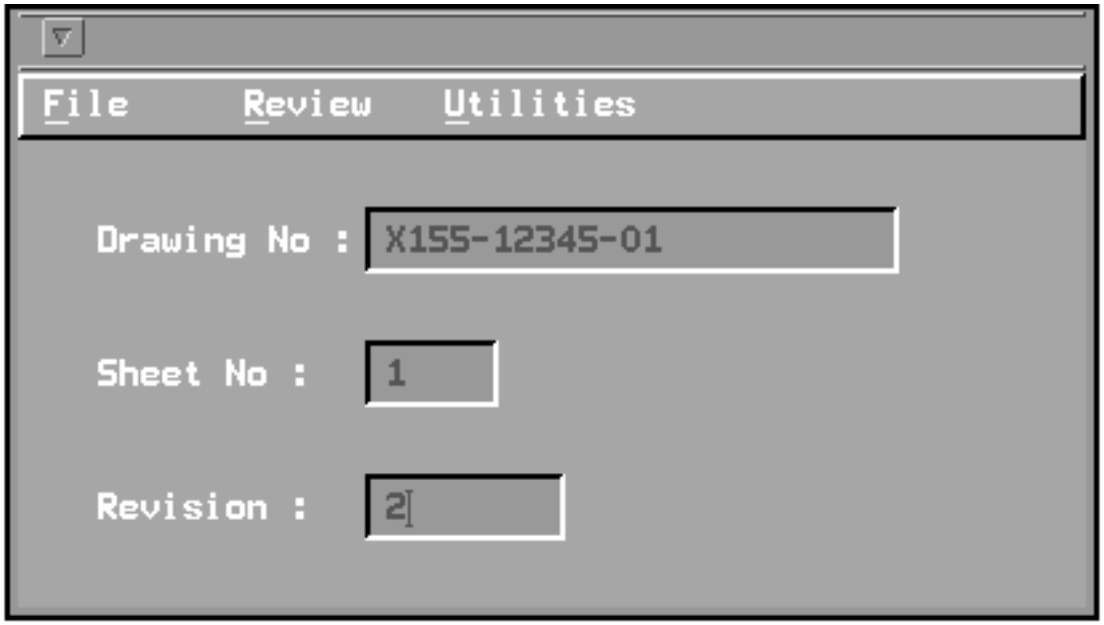
Please note: The name of this second window is defined against the `QPOPUP` command in the main resource file and not within the resource file defining the second window.

Example of a Menu Driven Interface

The previous examples show interfaces driven by user input fields and on-screen buttons. This is the best approach for very simple applications since all possible actions the user can take are clearly displayed. However, a menu-driven style interface is more suitable for larger applications.

The following figure shows a possible drawings access system, which provides file access facilities, drawing review functions, and various utilities.

Figure 7-9 Example of a Menu Driven Interface



The following example shows how to create this menu structure.

Definition of the Main Window

The main window of this application enables a user to specify a drawing number, sheet number, and revision, and then select an option from one of the three menus available.

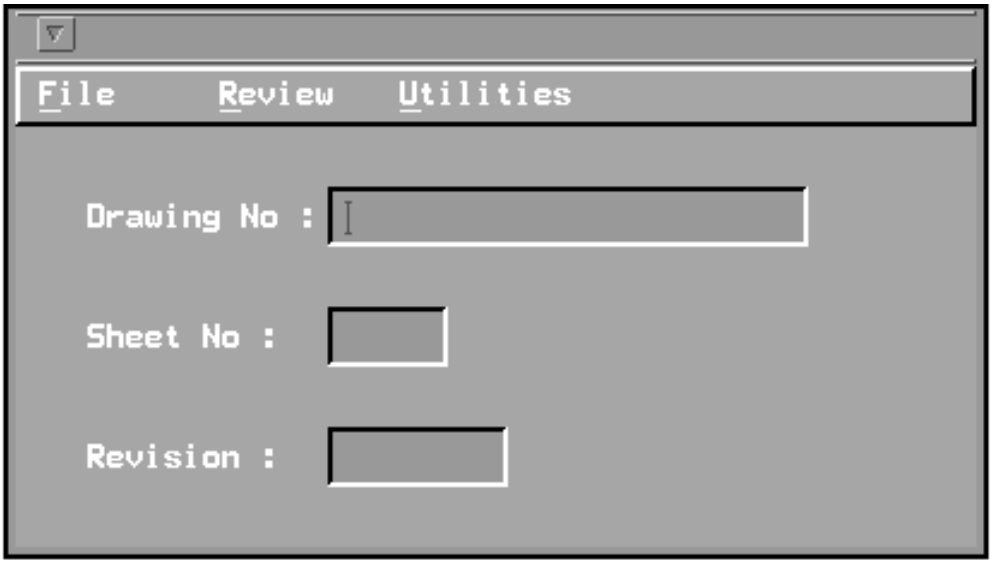
The Complete Resource File

The basic resource file to define the window, input fields and menu bar is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,200,LABEL,Main Window,QEND
# ================================================
# Define the menu bar
QPANEL
XPOS,0,YPOS,0,WIDTH,400,HEIGHT,25,QEND
QMESSAGE
XPOS,10,YPOS,6,WIDTH,75,HEIGHT,25,LABEL,File,ACTION,MU,filemenu,Q
END
QMESSAGE
XAFTER,0,YOFFSET,0,WIDTH,75,HEIGHT,25, ->
LABEL,Review,ACTION,MU,revmenu,QEND
QMESSAGE
XAFTER,0,YOFFSET,0,WIDTH,75,HEIGHT,25, ->
LABEL,Utilities,ACTION,MU,utilmenu,QEND
# ================================================
# Drawing Number input text field
QTEXT
QID,dwg_no,XPOS,130,YPOS,50,WIDTH,200,HEIGHT,25,INVERT,1, ->
QEND
QMESSAGE
XPOS,80,YPOS,60,LABEL,Drawing NO :,QEND
# Sheet Number input text field
QTEXT
QID,sheet_no,XPOS,130,YPOS,100,WIDTH,50,HEIGHT,25,INVERT,1, ->
QEND
QMESSAGE
XPOS,80,YPOS,60,LABEL,Drawing NO :,QEND
# Drawing Revision input text field
QTEXT
QID,rev_no,XPOS,130,YPOS,150,WIDTH,75,HEIGHT,25,INVERT,1, ->
QEND
QMESSAGE
XPOS,80,YPOS,60,LABEL,Drawing NO :,QEND
```

This resource file definition creates the main window shown in the following figure.

Figure 7-10 Example of a Main Window



Defining a Menu Bar

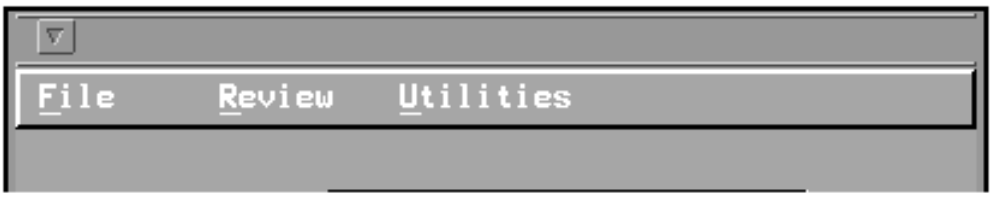
A menu bar can be defined using a QPANEL to make it stand out. The definitions for the menu bar are as follows:

Example:

```
QPANEL
XPOS,0,YPOS,0,WIDTH,400,HEIGHT,25,QEND
QMESSAGE
XPOS,10,YPOS,6,WIDTH,75,HEIGHT,25,LABEL,File,ACTION,MU,filemenu,Q
END
QMESSAGE
XAFTER,0,YOFFSET,0,WIDTH,75,HEIGHT,25, ->
LABEL,Review,ACTION,MU,revmenu,QEND
QMESSAGE
XAFTER,0,YOFFSET,0,WIDTH,75,HEIGHT,25, ->
LABEL,Utilities,ACTION,MU,utilmenu,QEND
```

The preceding definitions create the following menu bar.

Figure 7-11 Example of a Menu Bar



These QMESSAGE items reference QMENU items that must also be defined in the window resource file.

The QMENU definitions for the File menu, the Review menu and the Utilities menu follow:

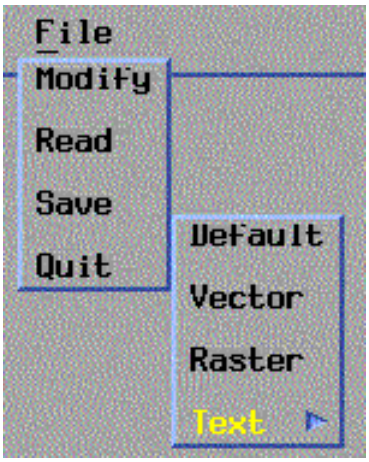
```
QMENU
QID,filemenu,COLS,8,ROWS,6,LISTFILE
,/usr/menus/myfilemenu,DEPTH,1,QEND
QMENU
QID,revmenu,COLS,8,ROWS,6,LISTFILE
,/usr/menus/myrevmenu,DEPTH,1,QEND
QMENU
QID,utilmenu,COLS,8,ROWS,6,LISTFILE
,/usr/menus/myutilmenu,DEPTH,1,QEND
```

These must be included in the window resource file definition.

Creating Sub-menu Options

Selecting the File menu option displays a menu with four options.

Figure 7-12 Example of Sub-menu option



Please note: The arrowhead against the Text option indicates that there are more sub-options available for this selection.

The contents of the File menu are held in a Menu Definition File named /usr/menus/myfilemenu, as identified by the QMENU item of the resource file.

```
Modify    1000
Read      1010
Save      MU,savemenu
Quit      EX
```

The format of a Menu Definition File comprises the text of the menu option followed by an action, separated by a TAB character. For more information see "Working with Menus" on page 3-35.

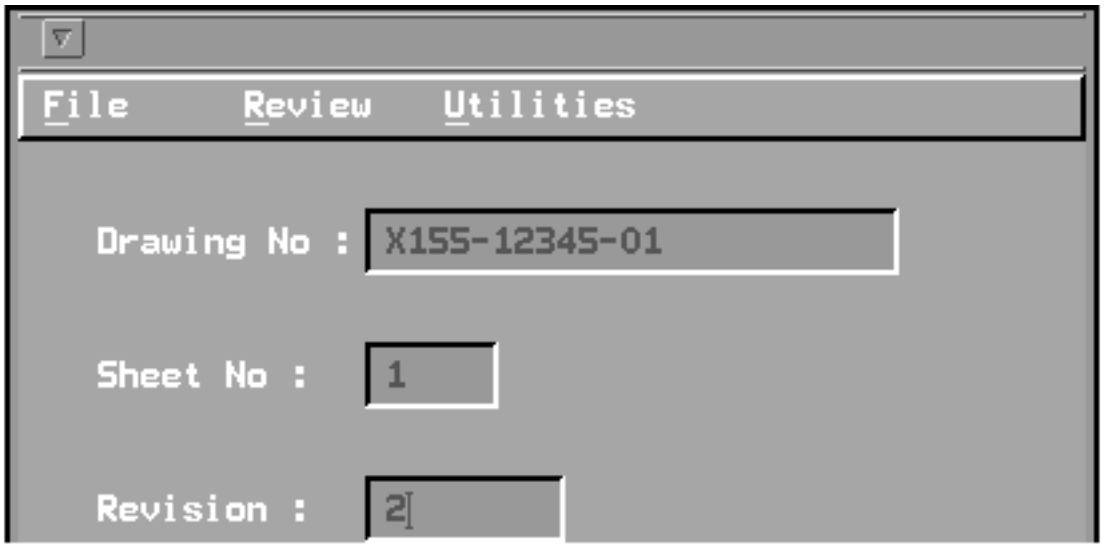
To invoke a sub-menu for Save, the option takes an AccessWare MU action argument to bring up the next menu, which must be defined as a QMENU item in the resource file.

The definition of the Save menu is as follows:

```
QMENU
QID,savemenu,COLS,8,ROWS,6,LISTFILE,/usr/menus/mysavemenu,->
DEPTH,1,QEND
```

Selecting the SAVE option creates the screen seen in the following figure:

Figure 7-13 Example of a Save window



The contents of the Save Menu Definition File /usr/menus/mysavemenu are as follows:

```
Default  1020
Vector   1030
Raster   1040
Text     MU,textmenu
```

The Text option has another AccessWare MU action to bring up the next menu, which must also be defined as a QMENU item in the resource file.

```
QMENU
QID,textmenu,COLS,8,ROWS,6,LISTFILE,/usr/menus/mytextmenu,DEPTH,1
,QEND
```

The nested sub-menu for the Text option is indicated by the arrowhead against the Text option.

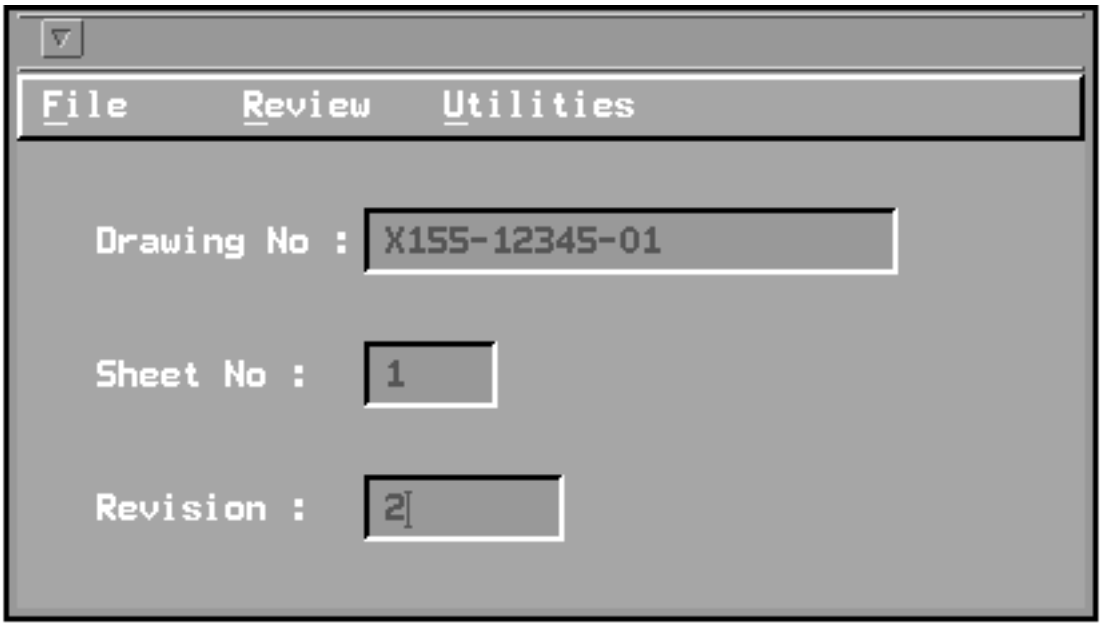
The Text menu file can contain actions as follows:

```
IGES     1050
DXF      1060
```

In the preceding Menu Definition Files, all the ACTIONs except the Menu ACTIONs and the Exit action are performed by an application program. The references 1000,1010,... 1060, identify the statements to be executed in the program.

Selecting the Save > Text option displays the full set of options from this sequence of walking menus as follows:

Figure 7-14 Example of a Text sub-menu window



The options in the Review and Utilities menus could be as follows:

The contents of the Review menu are held in a Menu Definition File named `/usr/menus/myrevmenu`, as identified in the `QMENU` item of the resource file.

A Review Menu Definition File may contain menu options followed by an action as follows:

```
Accept  SCR,accept_scr,dwg_no~sheet_no~rev~QQ
Reject  SH,reject_win
```

The contents of the Utilities menu are held in the file named `/usr/menus/myutilmenu`, as identified in the `QMENU` item of the resource file.

A Utilities Menu Definition File may contain options as follows:

```
Print   SCR,print_scr,dwg_no~sheet_no~rev~QQ
Mail    SH,mail_win
Query   CF,query_actions
Remove  SCR,del_scr,dwg_no~sheet_no~rev~QQ
```

Please note: The use of script actions, command files and other AccessWare actions in these files.

Menu Item Definition Files

For the preceding example, the complete set of Menu Definition Files required is as follows:

File: `/usr/menus/myfilemenu`:

```
Modify  1000
Read    1010
Save    MU,savemenu
Quit    EX
```

File: `/usr/menus/mysavemenu`:

```
Default 1020
Vector  1030
Raster  1040
Text    MU,textmenu
```

File: `/usr/menus/mytextmenu`:

```
IGES    1050
DXF     1060
```

File: /usr/menus/myrevmenu:

```
Accept  SCR,accept_scr,dwg_no~sheet_no~rev~QQ
Reject  SH,reject_win
```

File: /usr/menus/myutilmenu:

```
Print   SCR,print_scr,dwg_no~sheet_no~rev~QQ
Mail    SH,mail_win
Query   CF,query_actions
Remove  SCR,del_scr,dwg_no~sheet_no~rev~QQ
```

The menus created by the preceding menu definitions files are displayed as follows:

Figure 7-15 Example of Sub-menu options

The Complete Resource File

The complete resource file definition for this example is as follows:

```
QWINDOW
XLOC,100,YLOC,200,WIDTH,400,HEIGHT,200,LABEL,Main Window,QEND
# ================================================
# Define the menu bar
QPANEL
XPOS,0,YPOS,0,WIDTH,400,HEIGHT,25,QEND
QMESSAGE
XPOS,10,YPOS,6,WIDTH,75,HEIGHT,25,LABEL,File,ACTION,MU,->
filemenu,QEND
QMESSAGE
XAFTER,0,YOFFSET,0,WIDTH,75,HEIGHT,25, ->
LABEL,Review,ACTION,MU,revmenu,QEND
QMESSAGE
XAFTER,0,YOFFSET,0,WIDTH,75,HEIGHT,25, ->
LABEL,Utilities,ACTION,MU,utilmenu,QEND
# ================================================
# Define the input fields
# Drawing Number input text field
QTEXT
QID,dwg_no,XPOS,130,YPOS,50,WIDTH,200,HEIGHT,25,INVERT,1, ->
QEND
```

```
QMESSAGE
XPOS,80,YPOS,60,LABEL,Drawing NO :,QEND
# Sheet Number input text field
QTEXT
QID,sheet_no,XPOS,130,YPOS,100,WIDTH,50,HEIGHT,25,INVERT,1, ->
QEND
QMESSAGE
XPOS,80,YPOS,60,LABEL,Sheet NO:,QEND
# Drawing Revision input text field
QTEXT
QID,rev_no,XPOS,130,YPOS,150,WIDTH,75,HEIGHT,25,INVERT,1, ->
QEND
QMESSAGE
XPOS,80,YPOS,60,LABEL,Revision NO :,QEND
# ================================================
# Define the menus
# The filemenu
QMENU
QID,filemenu,COLS,8,ROWS,6,LISTFILE,/usr/menus/myfilemenu,->
DEPTH,1,QEND
# The reviewmenu
QMENU
QID,revmenu,COLS,8,ROWS,6,LISTFILE,/usr/menus/myrevmenu,->
DEPTH,1,QEND
# The utilities menu
QMENU
QID,utilmenu,COLS,8,ROWS,6,LISTFILE,/usr/menus/myutilmenu,->
DEPTH,1,QEND
# The savemenu
QMENU
QID,savemenu,COLS,8,ROWS,6,LISTFILE,/usr/menus/mysavemenu,->
DEPTH,1,QEND
# The text menu
QMENU
QID,textmenu,COLS,8,ROWS,6,LISTFILE,/usr/menus/mytextmenu,->
DEPTH,1,QEND
# ================================================
# End of window definition
QATTREND
# End of window definition
#
```

Please note: The actions in the Review and Utilities Menu Definition Files refer to two other windows with QID's of reject_win and mail_win. In a complete application, these would also be defined as QPOPUPs in this resource file.

Example of Extended Menus

AccessWare supports extended menus. You can define extended menus by defining the QMENUBAR item in your resource file as follows:

```
QMENUBAR
STATE,STATIC,QID,mymenubar,XPOS,0,YPOS,0,WIDTH,500,HEIGHT,15,->
ITEMFILE,/home/demo/menubar,QEND
```

Menubar Definition File

The following is a sample menubar definition file:

```
&File         $CA_MENUS/../file.men
&Edit         $CA_MENUS/../edit.men
&View         $CA_MENUS/../view.men
&Database     $CA_MENUS/../database.men
&Tools        $CA_MENUS/../tools.men
&Display      $CA_MENUS/../display.men
&Window       $CA_MENUS/../windows.men
&Admin        $CA_MENUS/../admin.men
&Help         $CA_MENUS/../help.men
```

Menu Item Definition File

The following is a sample menu item file `edit.men`. This file contains the QIDs of the menu items whose variable value is displayed in the Edit menu on the menubar. Each menu item is labelled and is defined by the keyword `EXTENDED`.

Please note: For ordinary menus the keyword `EXTENDED` is not required.

```
CA_MENUITEM_EDIT_ADDCOMPONENTAdd ComponentEXTENDED
CA_MENUITEM_EDIT_SWAPCOMPONENTSwap ComponentEXTENDED
CM_MENUID_ADDAddEXTENDED
CM_MENUID_CHANGEChangeEXTENDED
CM_MENUID_REMOVERemoveEXTENDED
```

Please note: All fields in both the Menubar Definition File and the Menu Item Definition File are separated by `TAB` spaces.

In the preceding example, consider the following menu item:

```
CA_MENUITEM_EDIT_ADDCOMPONENT
```

This string is defined by the QMENUITEM item in the file menus.r as follows:

```
QMENUITEM
QID,CA_MENUITEM_EDIT_ADDCOMPONENT,->
LABEL,$CA_LABEL_EDIT_ADDCOMPONENT,->
VISIBLEDEP,CA_EXPR_STD_OR_SAP,ACTION,7830,QEND
```

In the preceding definition, consider the following:

- The VISIBLEDEP,CA_EXPR_STD_OR_SAP value represents the control state of the menu item CA_MENUITEM_EDIT_ADDCOMPONENT. This value is defined in a separate file states.r as follows:

  ```
  QEXPRESSION
  QID,CA_EXPR_STD_OR_SAP,VALUE,1,->
  RULES,CA_STATE_STD_MODE || CA_STATE_SAP_MODE,QEND
  ```

- The QCONTROLSTATE item definitions for CA_STATE_STD_MODE and CA_STATE_SAP_MODE are as follows:

  ```
  QCONTROLSTATE
  QID,CA_STATE_STD_MODE,VALUE,1,QEND
  QCONTROLSTATE
  QID,CA_STATE_SAP_MODE,VALUE,0,QEND
  ```

Please note: The file names need not have the extension given in this example.

Appendix A

Attribute Cross-Reference Table

This appendix provides a cross-reference table of attributes applicable to items. Use the table to determine the required attributes in AccessWare.

- Attribute Cross-Reference Table

Attribute Cross-Reference Table

Table A-1 Attributes Cross-referenced to Items

	QWINDOW	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
ACTION	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y			Y		
ACTIVE		Y	Y	Y	Y	Y	Y	Y			Y	Y					
ACTIVEDEP															Y		
BACKCOLOR		Y	Y	Y	Y	Y	Y	Y									
COLS		Y	Y	Y	Y	Y	Y	Y									
DEPTH		Y	Y	Y	Y		Y	Y			Y			Y			
DICON			Y	Y													
ETCH		Y	Y	Y													
FONT	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y			
GLOBVARS	Y																
GREYOUTCOL		Y	Y	Y	Y	Y	Y							Y			
HEIGHT	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y			
HELPFILE		Y	Y	Y	Y	Y	Y				Y	Y		Y			
HELPWIN	Y																
ICON	Y			Y													
ICONLABEL	Y																
INVERT		Y	Y	Y			Y										
ITEMCOLOR		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
LABEL	Y	Y		Y	Y	Y	Y		Y		Y	Y			Y		
LAYOUT					Y							Y					
LISTFILE			Y					Y		Y			Y				
LISTMENU			Y														
MODE			Y		Y		Y										
NOBANNER	Y																
NOSTRIPE	Y																
PALETTE	Y																
PROMPT		Y	Y	Y	Y	Y	Y				Y	Y					

Table A-1 Attributes Cross-referenced to Items

	QWINDOW	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
QID		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
ROWS		Y	Y				Y	Y									
RULES																	Y
SELECTCOLOR			Y	Y	Y			Y									
STATE			Y	Y			Y									Y	
STYLE				Y	Y												
TEXTCOLOR		Y	Y	Y	Y	Y	Y	Y									
UDATA															Y		
VALUE		Y		Y	Y		Y				Y	Y				Y	Y
VISIBLE		Y	Y	Y	Y	Y	Y		Y		Y	Y		Y			
VISIBLEDEP															Y		
WIDTH	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y			
XAFTER		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
XLOC	Y																
XOFFSET		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
XPOS	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y			
YAFTER		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
YLOC	Y																
YOFFSET		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
YPOS	Y	Y	Y	Y	Y	Y	Y	Y			Y	Y	Y	Y			
ITEMFILE			Y					Y		Y			Y				
OVERVIEW			Y														
MOVEX		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
MOVEY		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
STRETCHX		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
STRETCHY		Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
RESIZE	Y																
SPLASH	Y																
INTERVAL												Y					

Table A-1 Attributes Cross-referenced to Items

	QWINDOW	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
MAXVALUE												Y					
FOCUS		Y	Y	Y	Y		Y	Y			Y	Y	Y	Y	Y		
DEFAULT				Y													
MAXWIDTH							Y										
MAXHEIGHT							Y										
MAXROWS							Y										
MAXCOLS							Y										
MINVALUE												Y					
MINWIDTH	Y																
MINHEIGHT	Y																
TIP				Y													

Appendix B

Function Cross-Reference Table

This appendix provides a cross-reference table of functions applicable to items. Use the table to determine the required functions in AccessWare.

- Function Cross-Reference Table

Function Cross-Reference Table

Table B-1 Functions Cross-referenced to Items

	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
aw_activate_item	Y	Y	Y	Y	Y	Y			Y	Y	Y		Y			
aw_add_item_to_group	Y	Y	Y	Y	Y	Y				Y						
aw_add_line_to_list		Y														
aw_beep																
aw_clear_message																
aw_clear_prompt										Y						
aw_close_window								Y								
aw_create_cgm_file		Y														
aw_cursor_into_item	Y	Y	Y	Y	Y	Y					Y		Y			
aw_deactivate_item	Y	Y	Y	Y	Y	Y			Y	Y	Y		Y			
aw_delete_line_from_list		Y														
aw_del_item_from_group	Y	Y	Y	Y	Y	Y										
aw_exit_macro										Y						
aw_exit_windows										Y						
*aw_get_active_id																
aw_get_cursor_position																
aw_get_global																
aw_get_item_action	Y	Y	Y	Y	Y	Y	Y				Y		Y	Y		
aw_get_item_active	Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
aw_get_item_backcolor	Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
aw_get_item_bit		Y		Y												
*aw_get_item_caret						Y										
aw_get_item_cols	Y	Y	Y	Y	Y	Y	Y									
aw_get_item_depth	Y	Y	Y	Y		Y	Y									
*aw_get_item_dicon			Y													
aw_get_item_dimension	Y	Y	Y	Y	Y	Y					Y	Y	Y			
aw_get_item_etch	Y	Y	Y													

Table B-1 Functions Cross-referenced to Items

	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
*aw_get_item_font	Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
aw_get_item_greycolor	Y	Y	Y	Y	Y	Y				Y						
aw_get_item_height	Y	Y	Y	Y	Y						Y		Y			
*aw_get_item_helpfile	Y	Y	Y	Y	Y	Y					Y		Y			
*aw_get_item_icon			Y													
aw_get_item_integer						Y										
aw_get_item_invert	Y	Y	Y			Y							Y			
aw_get_item_itemcolor	Y	Y	Y	Y	Y	Y	Y									
*aw_get_item_label	Y		Y	Y	Y	Y		Y			Y			Y		
aw_get_item_lowercase						Y				Y						
aw_get_item_mixedcase						Y				Y						
aw_get_item_multiselect		Y		Y												
aw_get_item_position	Y	Y	Y	Y	Y	Y				Y	Y		Y			
*aw_get_item_prompt	Y	Y	Y	Y	Y	Y				Y	Y		Y			
aw_get_item_readonly						Y										
aw_get_item_real						Y				Y						
aw_get_item_rows	Y	Y				Y				Y						
aw_get_item_textcolor	Y	Y	Y	Y	Y	Y	Y			Y						
aw_get_item_uppercase						Y										
*aw_get_item_value	Y		Y	Y		Y				Y	Y		Y		Y	Y
aw_get_item_visible	Y	Y	Y	Y	Y	Y		Y		Y	Y		Y			
aw_get_item_width	Y	Y	Y	Y	Y	Y	Y				Y		Y			
aw_get_item_xposition	Y	Y	Y	Y	Y	Y					Y		Y			
aw_get_item_yposition	Y	Y	Y	Y	Y	Y				Y	Y		Y			
*aw_get_line		Y					Y			Y						
*aw_get_list_file		Y					Y			Y		Y				
*aw_get_list_heading		Y								Y						
aw_get_list_length		Y					Y									
*aw_get_list_line		Y					Y									

Table B-1 Functions Cross-referenced to Items

	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
aw_get_list_width		Y					Y									
*aw_get_list_selection		Y														
aw_get_menuitem_state														Y		
aw_get_node_access		Y								Y						
aw_get_node_highlight		Y														
aw_get_node_linkcolor		Y														
aw_get_node_selected		Y														
aw_get_node_textcolor		Y														
aw_get_node_visible		Y														
aw_get_scroll_char		Y														
aw_get_scroll_line		Y														
aw_get_state															Y	
aw_get_text_integer						Y										
aw_get_text_lowercase						Y										
aw_get_text_mixedcase						Y										
aw_get_text_readonly						Y										
aw_get_text_real						Y										
aw_get_text_uppercase						Y										
*aw_get_udata														Y		
aw_get_window_dimension								Y								
aw_get_window_font								Y								
aw_get_window_height								Y								
aw_get_window_position								Y								
aw_get_window_resource								Y								
aw_get_window_stripe								Y								
aw_get_window_width								Y								
aw_hide_all_popups								Y		Y	Y		Y			
aw_hide_item	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y		Y			
aw_init_windows																

Table B-1 Functions Cross-referenced to Items

	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
aw_is_item_active	Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
aw_is_item_bit_set		Y		Y												
aw_is_item_multiselect		Y		Y		Y										
aw_is_item_readonly						Y										
aw_is_item_visible	Y	Y	Y	Y	Y	Y		Y		Y	Y	Y	Y			
aw_is_node_highlighted		Y														
aw_is_node_selected		Y														
aw_is_node_visible		Y														
aw_mark_list_line		Y														
aw_message																
aw_open_window								Y								
aw_perl_exec																
aw_prompt																
aw_refresh_list		Y														
aw_refresh_window								Y								
aw_register_repository																
aw_register_repository_latest																
aw_reset_item_bit		Y		Y												
aw_reset_list		Y														
aw_save_listfile		Y														
aw_scroll_to_char		Y														
aw_scroll_to_line		Y														
*aw_search_path																
aw_set_global																
aw_set_item_action	Y	Y	Y	Y	Y	Y		Y		Y	Y		Y	Y		
aw_set_item_active	Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
aw_set_item_backcolor	Y	Y	Y	Y	Y	Y	Y									
aw_set_item_bit		Y		Y												
aw_set_item_caret						Y										

Table B-1 Functions Cross-referenced to Items

	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
aw_set_item_cols	Y	Y	Y	Y	Y	Y	Y									
aw_set_item_depth	Y	Y	Y	Y		Y	Y									
aw_set_item_dicon			Y							Y	Y		Y			
aw_set_item_etch	Y	Y	Y							Y	Y		Y			
aw_set_item_font	Y	Y	Y	Y	Y	Y	Y									
aw_set_item_greycolor	Y	Y	Y	Y	Y	Y				Y	Y		Y			
aw_set_item_heading		Y								Y	Y		Y			
aw_set_item_height	Y	Y	Y	Y		Y										
aw_set_item_helpfile	Y	Y	Y	Y	Y	Y										
aw_set_item_icon			Y													
aw_set_item_integer						Y										
aw_set_item_invert	Y	Y	Y			Y				Y	Y		Y			
aw_set_item_itemcolor	Y	Y	Y	Y	Y	Y	Y									
aw_set_item_label	Y		Y	Y	Y	Y		Y		Y	Y		Y			
aw_set_item_lowercase						Y										
aw_set_item_mixedcase						Y										
aw_set_item_multiselect		Y		Y												
aw_set_item_position	Y	Y	Y	Y	Y	Y				Y	Y		Y			
aw_set_item_prompt	Y	Y	Y	Y	Y	Y				Y	Y		Y			
aw_set_item_readonly						Y				Y	Y		Y			
aw_set_item_real						Y				Y	Y		Y			
aw_set_item_rows	Y	Y				Y										
aw_set_item_selectcolor		Y	Y	Y			Y			Y	Y		Y			
aw_set_item_textcolor	Y	Y	Y	Y	Y	Y	Y			Y	Y		Y			
aw_set_item_uppercase						Y				Y	Y		Y			
aw_set_item_value	Y		Y	Y		Y				Y	Y		Y			
aw_set_item_visible	Y	Y	Y	Y	Y	Y		Y		Y	Y		Y			
aw_set_item_width	Y	Y	Y	Y		Y										
aw_set_item_xposition	Y	Y	Y	Y	Y	Y										

Table B-1 Functions Cross-referenced to Items

	QPANEL	QLIST	QBUTTON	QCHOICE	QMESSAGE	QTEXT	QMENU	QPOPUP	QGROUP	QTABCONTROL	QSLIDER	QMENUBAR	QSEPARATOR	QMENUITEM	QCONTROLSTATE	QEXPRESSION
aw_set_item_yposition	Y	Y	Y	Y	Y	Y										
aw_set_list_file		Y					Y					Y				
aw_set_list_heading		Y														
aw_set_node_access		Y														
aw_set_node_highlight		Y														
aw_set_node_linkcolor		Y														
aw_set_node_selected		Y														
aw_set_node_visible		Y														
aw_set_state															Y	
aw_set_text_integer						Y										
aw_set_text_lowercase						Y										
aw_set_text_mixedcase						Y										
aw_set_text_readonly						Y										
aw_set_text_real						Y										
aw_set_window_height								Y								
aw_set_window_position								Y								
aw_set_window_resource								Y								
aw_set_window_width								Y								
aw_show_item	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y		Y			
aw_show_item_relative	Y		Y	Y	Y	Y		Y		Y	Y		Y			
aw_start_macro																
aw_switch_list_font		Y														
aw_system																
aw_to_cadds																
aw_update_list		Y														
aw_win_message								Y								
aw_win_prompt								Y								

Please note that functions not cross-referenced to any item are for general usage in the application.

Glossary

This section explains the terms used in this document.

ACTION

The AccessWare keyword that in conjunction with a set of arguments initiates actions on the interface.

Actions

The generic term used in AccessWare to describe commands that cause the application code to produce a change or effect within the interface.

Action Attribute

The keyword and arguments that define the actions produced in the interface when an item is selected with the mouse or when a window opens or closes.

Button

A graphical representation on the screen of a button that produces an action when selected with the mouse.

CADDS

CAD/CAM product range.

Color Palette

A file in AccessWare that defines the 16 color shades that can be used within an interface.

Environment Variables

Variables set by the user at the operating system level that provide values for variables used by the application.

Graphical User Interface (GUI)

An interface for a main application program that presents the user with a graphical window display, and allows the user to interact with it to access the functions of the main application program.

Grey Out

The term used to denote a deactivated item, usually shown by 'greying-out' the item on the screen.

Help File

A text file giving Help information and displayed in a Help window.

Help Window

A window defined specifically to display a Help file.

Icon

A graphical image displayed in the window or on another window item to represent the item or window.

Item Attribute

An individual parameter defining the characteristics of a window or window element.

Menu

A list of displayed options that produces action on the interface when the user selects an option.

Menu Item Definition File

A text file listing the contents for a menu.

Panel

Term used in this manual to mean other rectangular areas displayed as a window or within the window.

Popup

Term used in AccessWare to describe a panel that ‘pops up’ when a mouse button is operated in another window element.

Tree

A way of presenting information graphically, which shows the structure hierarchically in the form of a tree. The nodes on the tree represent data items, and the links represent the relationships between the data items.

Programmatic Interface

A facility available in AccessWare allowing access to the user’s own programs to perform a series of actions in the interface.

QATTREND

The AccessWare keyword used to terminate the definition of a window’s resource file.

QBUTTON

The AccessWare keyword used to create an item that emulates a button within a window, and to introduce its definition in the resource file of the window in which it appears.

QCONTROLSTATE

The AccessWare keyword is used to define visibility or activity of a menu item.

QCHOICE

The AccessWare keyword used to display a set of options as button within a window, and to introduce its definition in the resource file of the window in which it appears.

QEND

The AccessWare keyword used to terminate the definition (the list of attributes and arguments) of an AccessWare item in a resource file.

QEXPRESSION

The AccessWare keyword defines a boolean of two or more control states.

QGROUP

The AccessWare keyword used to identify a set of items grouped for common actions, and to introduce its definition in the resource file of the window from which it is invoked.

QID

The name that uniquely identifies the item and is used as its reference across an entire interface.

QLIST

The AccessWare keyword used to create a list of information within a window, and to introduce its definition in the resource file of the window in which it appears.

QMENU

The AccessWare keyword used to identify a menu definition file that presents a list of menu options within a window, and to introduce its definition in the resource file of the window from which it is invoked.

QMENUBAR

The AccessWare keyword used to define top level menus for an application.

QMENUITEM

The AccessWare keyword used to define a single menu item of a menu.

QMESSAGE

The AccessWare keyword used to create a single line text field within a window that does not allow user entry, and to introduce its definition in the resource file of the window in which it appears.

QPANEL

The AccessWare keyword used to create a rectangular area within a window, and to introduce its definition in the resource file of the window in which it appears.

QPOPUP

The AccessWare keyword used to identify another window (the link to another resource file where the new window is defined), and to introduce its definition in the resource file of the window from which it is invoked.

QSEPARATOR

The AccessWare keyword used to generate a separation within a panel or a window.

QSLIDER

The AccessWare keyword used to generate a slide bar to adjust a numeric value.

QQ

The code used in AccessWare to terminate a list of items within an argument to an attribute. For example, the labels to a `QCHOICE` item.

QTABCONTROL

The AccessWare keyword used to define tab controls for a window.

QTEXT

The AccessWare keyword used to create a single line text field within a window that allows user entry, and to introduce its definition in the resource file of the window in which it appears.

QWINDOW

The AccessWare keyword used to identify the definition of a window in its resource file.

Resource File

A text file defining a window in an interface and all the various window elements within it.

Script Interface

A facility available in AccessWare allowing access to a script (a text file), containing the user's own commands written at the operating system level, which performs a series of actions in the interface.

Text Field

A field that appears within an AccessWare window or another window element that contains text entered by the user, the application, or defined as a label.

Window

A rectangular display area generated by the window system.

Window Element

Any item that appears within a window. For example, a button, a text field, or a menu.

Index

A

ACCESSTRACE 2-8
ACTION attribute 4-55
  action command file 4-57
    CF-action 4-57
  CADDS commands 4-66
    CAD1-send CADDS command without CR 4-67
    CAD-send CADDS command 4-66
  call scripts 4-64
    MACRO-define a macro 4-66
    SCR-run script 4-65
    SY-system call 4-64
    UL-update list 4-65
  coding actions 4-56
  form 4-55
  help facility 4-68
    HELP-toggle help 4-68
  invoke menus and popups 4-63
    MU-invoke menu 4-63
    POP-invoke popup 4-63
    POPREL-invoke and position a popup 4-64
  modification 4-59
    GI-grey in 4-59
    GO-grey out 4-59
    HI-hide 4-60
    SH-show 4-60
  operating windows 4-58
    CL-close window 4-58
    EX-exit windows 4-58
  scrolling QBUTTONs 4-68
  scrolling QLISTs 4-67
    HEAD-header bar 4-67
    HS-horizontal scrollbar 4-67
    VS-vertical scrollbar 4-68
  set values 4-60
    SV-set value 4-60
  special fields 4-61
    ME-message 4-62
    PR-prompt 4-62
ACTION Attribute Reference 4-54
ACTIVE attribute 4-25
ACTIVEDEP attribute 4-26
Application attributes 4-50
  GLOBVARS 4-52
  NOBANNER 4-51
  NOSTRIPE 4-51
  PALETTE 4-52
Application Code 6-4
  aw_user_exit procedure 6-4
  main procedure 6-4
AW_BACK_COLOR 2-6
AW_HILITE_COLOR 2-7
AW_ITEM_COLOR 2-6
AW_LABEL_COLOR 2-6
AW_LOLITE_COLOR 2-7
AW_MESSAGE_COLOR 2-6
AW_MESSAGE_FILE 2-8
AW_PROMPT_COLOR 2-6
AW_RESOURCE 2-5
AW_SCROLL_FACTOR 2-7
AW_SELECT_COLOR 2-6
AW_TEXT_COLOR 2-7
AW_WINDOW_STATE 2-8
AW_XICON_TYPE 2-8

B

BACKCOLOR attribute 4-33

C

CAD action 4-66
CAD1 action 4-67
CADDS commands actions 4-66
  CAD1-send CADDS command without CR 4-67
  CAD-send CADDS command 4-66
CF action 4-57
CGM file, creating 6-18
CL action 4-58
CMOM_DISPLAY 2-9
CMOM_DOMAIN 2-9
Color controlling attributes 4-33
  BACKCOLOR 4-33
  GREYOUTCOL 4-34
  ITEMCOLOR 4-33
  SELECTCOLOR 4-33
  TEXTCOLOR 4-33
Colors 3-43
  creating color palette file 3-43
  defining color palette 3-43
COLS attribute 4-24
Compiling and linking program 6-10
Concepts 1-4
  application actions 1-5
  color, icons and help 1-5
  resource files 1-4
creating a CGM file 6-18
Creating a user interface 3-5
  guidelines 3-2

D

Default settings attribute 4-39
  VALUE 4-39
DEPTH attribute 4-27
Development environment, setting 2-3
DICON attribute 4-31
Documentation, printing from Portable Document Format (PDF) file xvii
Dropdown Menus 3-37

E

Environment Variables 2-4
  ACCESSTRACE 2-8
  AW_BACK_COLOR 2-6
  AW_HILITE_COLOR 2-7
  AW_ITEM_COLOR 2-6
  AW_LABEL_COLOR 2-6
  AW_LOLITE_COLOR 2-7
  AW_MESSAGE_COLOR 2-6
  AW_MESSAGE_FILE 2-8
  AW_PROMPT_COLOR 2-6
  AW_RESOURCE 2-5
  AW_SCROLL_FACTOR 2-7
  AW_SELECT_COLOR 2-6
  AW_TEXT_COLOR 2-7
  AW_WINDOW_STATE 2-8
  AW_XICON_TYPE 2-8
  CMOM_DISPLAY 2-9
  CMOM_DOMAIN 2-9
  summary 2-4
ETCH attribute 4-28
EX action 4-58
Executing a script file 5-5
Executing applications 2-10
Extended menus 3-35
  example 7-20
  menubar definition file 3-37

F

Field contents attributes 4-37
  LISTFILE 4-37
  LISTMENU 4-38
FONT attribute 4-46

G

GI action 4-59
GLOBVARS attribute 4-52
GO action 4-59
GREYOUTCOL attribute 4-34
Guidelines, creating a user interface 3-2

H

HEAD action 4-67
HEIGHT 4-23
HELP action 4-68
Help facilities attributes 4-48
  HELPFILE 4-49
  HELPWIN 4-49
  PROMPT 4-48
  TIP 4-50
Help facility actions 4-68
  HELP-toggle help 4-68
HELPFILE attribute 4-49
HELPWIN attribute 4-49
HI action 4-60
HS action 4-67

I

ICON attribute 4-31
ICONLABEL attribute 4-32
Icons 3-40
  use of icons 3-40
    QBUTTON items 3-40
    QWINDOW items 3-40
Installation 2-2
  development environment, setting 2-3
  environment variables, setting 2-4
  runtime environment, setting 2-3
  system contents 2-2
INVERT attribute 4-27
Invoking ACTIONs 4-54
Item attributes 4-16
  assigning labels 4-35
    LABEL 4-35
  controlling color 4-33
    BACKCOLOR 4-33
    GREYOUTCOL 4-34
    ITEMCOLOR 4-33
    SELECTCOLOR 4-33
    TEXTCOLOR 4-33
  defining size 4-23
    COLS 4-24
    HEIGHT 4-23
    MAXCOLS 4-25
    MAXHEIGHT 4-24
    MAXROWS 4-25
    MAXWIDTH 4-24
    MINHEIGHT 4-23
    MINWIDTH 4-23
    ROWS 4-24
    WIDTH 4-23
defining visual style 4-25
    ACTIVE 4-25
    ACTIVEDEP 4-26
    DEPTH 4-27
    DICON 4-31
    ETCH 4-28
    ICON 4-31
    ICONLABEL 4-32
    INVERT 4-27
    LAYOUT 4-30
    STYLE 4-29
    VISIBLE 4-26
    VISIBLEDEP 4-27
field contents 4-37
    LISTFILE 4-37
    LISTMENU 4-38
help facilities 4-48
    HELPFILE 4-49
    HELPWIN 4-49
    PROMPT 4-48
    TIP 4-50
initial item defaults 4-39
    VALUE 4-39
miscellaneous 4-46
    FONT 4-46
    RULES 4-47
    UDATA 4-47
naming items 4-18
    QID 4-19
positioning items 4-19
    XAFTER 4-22
    XLOC 4-19
    XOFFSET 4-21
    XPOS 4-20
    YAFTER 4-22
    YLOC 4-19
    YOFFSET 4-21
    YPOS 4-20
resizing items 4-25
    MOVEX 4-25

MOVEY 4-25
STRETCHX 4-25
STRETCHY 4-25
special characteristics 4-41
MODE 4-43
STATE 4-41
whole application 4-50
GLOBVARS 4-52
NOBANNER 4-51
NOSTRIPE 4-51
PALETTE 4-52
Item modifying actions 4-59
GI-grey in 4-59
GO-grey out 4-59
HI-hide 4-60
SH-show 4-60
ITEMCOLOR attribute 4-33

L

LABEL attribute 4-35
Labelling attribute 4-35
LABEL 4-35
LAYOUT attribute 4-30
libraries, linking 6-3
UNIX 6-3
Windows NT 6-3
linking to libraries 6-3
UNIX 6-3
Windows NT 6-3
List contents, updating 5-8
LISTFILE attribute 4-37
LISTMENU attribute 4-38
Lists 3-27
data selection 3-30
multiple item selection mode 3-31
popup menu 3-30
selected row 3-30
single item selection mode 3-30
defining list contents 3-27
list headers 3-27
scrolling lists 3-28
scroll buttons 3-29
scrollbars 3-28
Lists and choice items, example 7-6

M

MACRO action 4-66
MAXCOLS attribute 4-25
MAXHEIGHT attribute 4-24
MAXROWS attribute 4-25
MAXWIDTH attribute 4-24
ME action 4-62
Menu driven interface 7-11
creating sub-menu options 7-14
defining a menu bar 7-13
defining main window 7-11
example 7-11
Menu invoking actions 4-63
MU-invoke 4-63
Menus 3-35
extended menus 3-35
example 7-20
menubar definition file 3-37
invoking a menu 3-37
popup menus 3-39
pulldown and dropdown 3-37
walking menus 3-38
ordinary menus 3-35
example 7-17
menu item definition file 3-35
MINHEIGHT attribute 4-23
MINWIDTH attribute 4-23
Miscellaneous attributes 4-46
FONT 4-46
RULES 4-47
UDATA 4-47
MODE attribute 4-43
MOVEX attribute 4-25
MOVEY attribute 4-25
MU action 4-63

N

Naming items attribute 4-18
QID 4-19
NOBANNER attribute 4-51
NOSTRIPE attribute 4-51

O

Ordinary Menus 3-35

P

PALETTE attribute 4-52
POP action 4-63
POPREL action 4-64
Popup invoking actions 4-63
  POP-invoke popup 4-63
  POREL-invoke and position a popup 4-64
Popup Menus 3-39
Positioning items attributes 4-19
  XAFTER 4-22
  XLOC 4-19
  XOFFSET 4-21
  XPOS 4-20
  YAFTER 4-22
  YLOC 4-19
  YOFFSET 4-21
  YPOS 4-20
PR action 4-62
Printing documentation from Portable Document Format (PDF) file xvii
Procedure Library 6-11
  item referencing 6-11
  summary of procedure calls 6-12
Programmatic Interface 6-2
  applying an ACTION 6-3
Programming examples 6-20
  creating confirmation panel 6-28
  passing arguments 6-20
  updating a list 6-25
  updating a text field 6-23
PROMPT attribute 4-48
Pulldown Menus 3-37

Q

QATTREND Keyword 4-15
QBUTTON Item 4-6
QBUTTON scrolling actions 4-68
QCHOICE Item 4-7
QCONTROLSTATE Item 4-12
QEXPRESSION Item 4-12
QGROUP Item 4-13
QID attribute 4-19
QLIST Item 4-9
  HEAD-header bar 4-67
  HS-horizontal scrollbar 4-67
  VS-vertical scrollbar 4-68
QMENU Item 4-10
QMENUBAR Item 4-15
QMENUITEM Item 4-11
QMESSAGE Item 4-8
QPANEL Item 4-5
QPOPUP Item 4-13
QSEPARATOR Item 4-15
QSLIDER Item 4-15
QTABCONTROL item 4-14
QTEXT Item 4-8
QWINDOW Item 4-4

R

Resizing items attributes 4-25
  MOVEX 4-25
  MOVEY 4-25
  STRETCHX 4-25
  STRETCHY 4-25
Resource file 3-5
  comment 3-11
  creating 3-10
  defining items 3-12
  example 1-6
  item ordering 3-12
  layout 3-11
  QWINDOW item 3-13
  structure 3-10
ROWS attribute 4-24
RULES attribute 4-47
Runtime environment, setting 2-3

S

SCR action 4-65
Script file, executing 5-5
Script Interface 5-2
  displaying messages 5-3
  passing arguments 5-4
  updating a text field 5-6

updating list contents 5-8
using 5-3
Script invoking actions 4-64
MACRO-define a macro 4-66
SCR-run script 4-65
SY-system call 4-64
UL-update list 4-65
SELECTCOLOR attribute 4-33
Setup procedures 2-1
SH action 4-60
Size and Position a window 4-4
Size defining attributes 4-23
COLS 4-24
HEIGHT 4-23
MAXCOLS 4-25
MAXHEIGHT 4-24
MAXROWS 4-25
MAXWIDTH 4-24
MINHEIGHT 4-23
MINWIDTH 4-23
ROWS 4-24
WIDTH 4-23
Special characteristics attributes 4-41
MODE 4-43
STATE 4-41
Special field actions 4-61
ME-message 4-62
PR-prompt 4-62
STATE attribute 4-41
STRETCHX attribute 4-25
STRETCHY attribute 4-25
STYLE attribute 4-29
SV action 4-60
SY action 4-64

T

Text field, updating 5-6
Text fields and buttons, example 7-2
TEXTCOLOR attribute 4-33
TIP attribute 4-50

U

UDATA attribute 4-47
UL action 4-65
Updating a text field 5-6
Updating list contents 5-8
User interface 3-5
examples 7-1
features 3-8
color palettes 3-8
Help 3-9
Messages 3-9
Prompts 3-9
use of icons 3-8
resource file 3-5
windows items 3-5

V

VALUE attribute 4-39
Value setting actions 4-60
SV-set value 4-60
VISIBLE attribute 4-26
VISIBLEDEP attribute 4-27
Visual style defining attributes 4-25
ACTIVE 4-25
ACTIVEDEP 4-26
DEPTH 4-27
DICON 4-31
ETCH 4-28
ICON 4-31
ICONLABEL 4-32
INVERT 4-27
LAYOUT 4-30
STYLE 4-29
VISIBLE 4-26
VISIBLEDEP 4-27
VS action 4-68

W

Walking Menus 3-38
WIDTH attribute 4-23
Window item reference 4-3
QATTREND keyword 4-15
QBUTTON item 4-6
QCHOICE item 4-7
QCONTROLSTATE item 4-12
QEXPRESSION item 4-12
QGROUP item 4-13
QLIST item 4-9
QMENU item 4-10

QMENUBAR item 4-15
QMENUITEM item 4-11
QMESSAGE item 4-8
QPANEL item 4-5
QPOPUP item 4-13
QSEPARATOR item 4-15
QSLIDER item 4-15
QTABCONTROL item 4-14
QTEXT item 4-8
QWINDOW item 4-4
Window operating actions 4-58
CL-close window 4-58
EX-exit windows 4-58

X

XAFTER attribute 4-22
XLOC attribute 4-19
XOFFSET attribute 4-21
XPOS attribute 4-20

Y

YAFTER attribute 4-22
YLOC attribute 4-19
YOFFSET attribute 4-21
YPOS attribute 4-20